WANG

VS

USERSUBS Reference

VS
USERSUBS Reference

1st Edition — August, 1982

800-1315US-01

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

Disclaimer of Warranties and Limitation of Liabilities

VS USERSUBS are made available by the International Society of Wang Users (ISWU) for the convenience of Wang customers. Customers choosing to use this software do so at their own risk. Neither ISWU nor Wang Laboratories, Inc., makes any representations regarding the accuracy of these programs or the associated documentation, and will not be responsible for any damages or loss of data arising from their use. Any comments or questions regarding VS USERSUBS should be directed to the International Society of Wang Users, Wang Laboratories, Inc., One Industrial Avenue, Lowell, Massachusetts.

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

PREFACE

This manual describes VS User Subroutines (USERSUBS), a collection of subroutines that provide programmers with system functions that are helpful in the development of application programs.

Chapter 1 provides a brief statement of the functions performed by each subroutine. Chapter 2 contains information about the subroutines and their descriptions, defines terms used in the manual, and provides instructions for coding the programming statements needed to access and use the subroutines. Chapter 3 contains the subroutine descriptions. Each description lists the subroutine's function(s), the argument list required to access it, notes on its use, and at least one example.

This manual is intended for a Wang VS user with programming experience. In particular, the programmer should be familiar with at least one supported programming language and should know how to reference subroutines in that language (although brief instructions are included in the manual). The programmer should also be familiar with the reference manual for the programming language being used, the *VS Programmer's Introduction*, and the *VS Program Development Tools*.

Use of some of these subroutines also requires an understanding of VS operating system details. The user should read, or be familiar with, the contents of both the *VS Operating System Services* and the *VS Principles of Operation*.

The reader should direct any comments about the documentation to Wang via the Customer Comment form inside the back cover of the manual.

The following VS manuals contain information useful to the programmer accessing USERSUBS:

Programmer's Introduction	800-1101PI
Principles of Operation	800-1100PO
Operating System Services	800-1107OS
Program Development Tools	800-1307PT
Procedure Language Reference	800-1205PR
Programming language references:	
Assembly Language Reference	800-1200AS
BASIC Language Reference	800-1202BA
COBOL Language Reference	800-1201CB
FORTRAN Language Reference	800-1208FR
PL/I Language Reference	800-1209PL
RPG II Language Reference	800-1203RP

CONTENTS

CONTENTS (continued)

TABLES

CHAPTER 1
SUBROUTINE FUNCTIONS

This chapter provides a brief statement of the function(s) of each user subroutine. This information is included in the subroutine descriptions in Chapter 3, but is summarized here for your convenience.

Subroutine	Function
BELL	Sounds the workstation alarm for a specified amount of time.
BITPACK	Converts a binary string into its ASCII character equivalent.
BITUNPK	Converts an ASCII character string into its binary equivalent.
CANCEL	Cancels execution of the calling program and displays a message on the workstation. The message consists of a message ID, a message issuer, and a message.
CEXIT	Overrides system cancel processing. On abnormal program termination, you can press PF1 to enter debug processing, PF16 to cancel processing, or the HELP key to access the Modified Command Processor. CEXIT allows you to restrict debug processing, to associate PF16 with alternate processing, and to disable operation of the HELP key.
CHKPARM	Performs table checking on one or more GETPARM keyword fields entered by a user or a procedure in a previous GETPARM request. You can use it for any type of field checking, but it is primarily intended for GETPARM Limited Alphanumeric and Alphanumeric keyword field types.
COMPRESS	Converts a character string to compressed format. Compressed format can reduce storage for records with repeated characters.
DATE	Performs the following date functions: (1) converts the current system date and time to a formatted string, (2) converts dates between Gregorian and Julian formats, (3) performs calculations with dates, and (4) determines the day of the week that corresponds to a given 20th century date.
DAY	Computes the day of the week that corresponds to a specified 20th century date.

Subroutine	Function
DISMOUNT	Initiates a dismount operation of a mounted disk or tape volume.
EXPAND	Converts a character string from compressed format to external format. EXPAND removes the special characters used to indicate repeated characters and produces text in noncompressed form.
EXTRACT	Provides information about the system and the program user.
FIND	Obtains one or more file, library, or volume names from complete or partial file, library, and volume names supplied by your program. Also indicates whether a specified file resides in a specified library and volume.
FLOPIO	Performs a variety of I/O operations on a nonlabeled (NL) diskette.
GETPARM	Enables you to generate parameter requests in a higher level language program.
HEXPACK	Converts a string of hexadecimal digits to its ASCII character equivalent.
HEXUNPK	Converts a string of ASCII characters into hexadecimal digits.
LINK	Allows your program to link to a program or procedure and to specify a cancel exit for the link. Your program can also specify any arguments that are needed to execute the linked program or procedure.
LOADCODE	Allows you to load specified microcode into a device.
LOGOFF	Terminates your program and logs you off the system.
MESSAGE	Allows communication of messages between workstations.
MOUNT	Allows you to mount a volume (disk or tape).
PAUSE	Causes a program to pause for a specified amount of time.
PRINT	Sends a print file to the print queue.
PROTECT	Changes the file security attributes of a file or library.
PUTPARM	Performs the following primary functions: (1) creates a parameter list (parameter reference block) to satisfy a subsequently generated parameter request, (2) retrieves a previously created parameter reference block, and (3) deletes existing parameter reference blocks.

Subroutine	Function
READFDR	Provides information about a specified file, including control blocks or file characteristics.
READVTOC	Provides information from the Volume Table of Contents (VTOC).
RENAME	Allows you to rename a file or library, with the options of bypassing expiration date checking and limiting access rights for a program with special privileges.
RETURN	Allows your program to return through several levels of subroutine calls.
SCRATCH	Provides the ability to scratch a file or library, with the options of bypassing expiration date checking and limiting access rights for a program with special privileges.
SEARCH	Performs a binary search on a specified table for a particular element and indicates whether the element exists in the table.
SET	Sets any of the allowable defaults that are available through the Command Processor SET Usage Constants function and the Procedure language SET command.
SORT	Sorts a character array on a specified field, in either ascending or descending order. Output from SORT can be either the sorted array or a locator-type array. (The elements in a locator-type array indicate the positions of the sorted elements in the character array.)
STRING	Provides the following string manipulation functions: (1) moves a string to another variable and pads it with a specified character, (2) moves a portion of a string to another variable, (3) centers a string, (4) left- or right-justifies a string, (5) reverses the order of characters in a string, and (6) translates the string according to a standard or user-specified translation table.
SUBMIT	Submits a background job to be run or held for later processing.
UNITRES	Allows you to reserve or release a device or peripheral processor on the system.
UPDATFDR	Allows you to update the VTOC entry of a file or library.
WSXIO	Performs I/O operations at the workstation and returns values associated with those operations.

CHAPTER 2
INTRODUCTION

2.1 PRELIMINARY INFORMATION

2.1.1 Why Use These Subroutines?

These subroutines provide very useful functions to the application programmer. Without them, it would frequently be necessary for you to know operating system details and how to program in Assembly language. These subroutines provide you with a simple means of accessing information that is not readily available.

2.1.2 Organization of Individual Subroutine Descriptions

As much as possible, individual subroutine descriptions are similar in format. Each description is divided into four sections: FUNCTION, USAGE, NOTES, and EXAMPLE. A description of each section follows.

FUNCTION — Mentions briefly the functions performed by the subroutine. After reading this section, you should know whether the subroutine is suitable for an intended application.

USAGE — Provides a general form of the argument list and a detailed discussion of the use of each argument. The following information is included.

Function: Some subroutines offer several different functions. When this is the case, a statement of each function appears before the applicable list of arguments.

Position: Indicates argument positions in the calling sequence.

Argument: Includes a descriptive name for each argument.

Type: Specifies the data type of the argument. Section 2.1.5 deals with data types.

Size: Indicates the number of bytes the argument must have.

Comments: Provides information about each argument, including its definition, restrictions on its use, and permissible values.

NOTES — Includes restrictions, precautions, programming hints, and other information about the subroutine.

EXAMPLES Illustrate the use of each subroutine. Most subroutines have examples written in COBOL; some in BASIC, RPG II, and FORTRAN. Examples were written and tested with the following compiler versions:

BASIC	3.03.01
COBOL	3.03.02
FORTRAN	2.05.00
RPG II	4.02.01

2.1.3 Conventions Used in the Manual

Argument List — The USAGE section of each subroutine description begins with a general argument list. Because subroutines have differing requirements, you can specify the argument list in a number of ways.

"arg1, ..., argn" means that there are n arguments, which you must specify in a particular order. The last argument in the general list indicates the maximum number of arguments that can be specified.

"argn, arguments" means that, for a subroutine that performs several functions, a particular argument ("argn") selects a function, which has its own argument requirements. Each function is described in detail.

"key1, rec1, ..." means that the program specifies arguments in "keyword-receiver" pairs. A keyword selects a particular option, and a receiver is associated with, and must be specified for, that keyword. The receiver can have a value that must be sent to the subroutine, or it can contain a value provided by the subroutine. Section 2.1.4 provides a definition of keyword and receiver.

Alpha n — Indicates that the data type for the argument is alphanumeric and that the number of bytes it must contain is n. "Var" indicates that the program can select the number of bytes or that the number depends on the information returned by the subroutine. Section 2.1.5 discusses alphanumeric data.

Integer 4 — Indicates that the data type of the argument is integer and that it must contain 4 bytes. This requirement presents a problem for COBOL programmers and is discussed in Section 2.2.2. Section 2.1.5 discusses integer data.

2.1.4 Terms Used in the Manual

AID Character — Indicates the workstation status (whether the keyboard is locked or unlocked) or which PF key the program operator pressed last. Table 3-18 is a complete list of AID (Attention ID) characters with their hexadecimal and ASCII character equivalents.

Argument List — Values (or locations where values can be obtained) required by the subroutine. It also includes variable names (or locations) that contain values returned by the subroutine. The CALL statement that references the subroutine typically contains an argument list.

CALL Statement	The statement that references the subroutine. It contains the CALL verb, the subroutine name, and an argument list. Each supported programming language uses a different form of CALL statement. Each is discussed in Section 2.2.
Character String	A sequence of alphanumeric characters, such as ABCDE or S#. These subroutines limit most character strings to letters and numbers, although some use special characters.
Keyword	Selects an option provided by the subroutine. For example, the SET subroutine allows you to set system parameters. For this subroutine, a keyword selects a parameter to be set.
Receiver	A variable that can be used to pass information to or receive information from a subroutine.
Return Code	Indicates whether or not the action requested by the program is successful. Many subroutines require that the program include an argument for a return code in the argument list. If the operation of the subroutine is successful, the value of the return code is zero. If unsuccessful, the return code corresponds to an error condition. For each subroutine that uses return codes, the subroutine description includes a table of codes and their meanings.
X'nn'	Hexadecimal representation for the value enclosed within quotes.

2.1.5 Data Types

All arguments contain data that is either alphanumeric or integer type. A discussion of both types follows.

Alphanumeric Data

Alphanumeric data consists of all characters in the character set, whether or not printable. Most alphanumeric subroutine arguments have values that are limited to uppercase letters and numbers, although some can use special characters and lowercase letters.

Each character of alphanumeric data requires one byte of storage. The Size section of each argument description provides the required size of the argument. An Alpha argument with size 8, for example, is an argument of eight alphanumeric characters requiring eight bytes of storage.

The various programming languages treat alphanumeric data differently. Section 2.2 explains each approach.

Integer Data

In these subroutines, all arguments having numeric values are integer type. Integers are whole numbers, expressed without fractional parts.

All arguments in these subroutines that contain integer values require four bytes of storage. The subroutines do not require integer (fullword) alignment.

Different programming languages have different ways of specifying and handling integer data. Section 2.2 discusses integer data.

2.2 HOW TO USE THE SUBROUTINES

First, select the appropriate subroutine from the brief description in Chapter 1 and the detailed description in Chapter 3.

Second, read the description of the subroutine and its arguments. Determine which values must be supplied by the program and which arguments contain values returned by the subroutine.

Third, add the necessary statements to the program to define argument values, call the subroutine, and use the values returned in arguments. Each programming language treats these statements differently. The necessary statements are described in the subsections that follow.

Fourth, run the program, first linking the external USERSUBS subroutine to it. Section 2.2.6 contains brief instructions on the use of the LINKER; refer to the *VS Program Development Tools* for more detailed information.

2.2.1 BASIC Language

Calling the Subroutine

The form of the BASIC language CALL statement for these subroutines is as follows:

CALL "subname" ADDR (arguments)

Subname is the name of the subroutine. The double quotation marks must be present in the CALL statement.

Arguments must be enclosed within parentheses and must be separated by commas. They must appear in the order specified in the argument list. In addition, each argument must agree in type and size with the corresponding argument in the list.

Alphanumeric Data

Variables with alphanumeric values must have names whose last character is the dollar sign ($). Alphanumeric constants are specified by enclosing their values within single or double quotes. (Note that single-quote literals provide lowercase letters.)

Example:

```
OPT$ = ''FC''
PROTECTCLASS$ = ''#''
CALL ''SET'' ADDR (OPT$,PROTECTCLASS$)

are equivalent to
CALL ''SET'' ADDR (''FC'',''#'')
```

Use the DIM statement to specify the names and number of characters of all alphanumeric variables.

Example:

```
DIM OPT$2, PROTECTCLASS$1
```

Integer Data

A variable with an integer value is designated as integer data type by appending the "%" character to its name. An integer constant that is defined in the user program contains a number followed by the "%" character. An integer that is input to the program via the workstation or a data file, computed in the program, or converted from an alphanumeric expression, is not followed by the "%" character. Integer data is stored in four bytes.

Example:

```
TIME% = 5%
CALL ''BELL'' ADDR (TIME%)

are equivalent to
CALL ''BELL'' ADDR (5%)
```

2.2.2 COBOL Language

Arguments passed to a subroutine must be defined in the Data Division. They can be initialized in either the Data Division or the Procedure Division.

Calling the Subroutine

The form of the CALL statement in COBOL is as follows:

CALL "subname" USING arguments

Subname is the subroutine name; it must be enclosed within quotes. Arguments are passed by means of the USING phrase. If the argument list for the subroutine specifies a certain order for the arguments, they must appear in that order in the USING phrase. Arguments that provide data to the subroutine must be variables that have been assigned values. Literals cannot be passed as arguments.

Alphanumeric Data

To define a data item as alphanumeric, its PICTURE character-string must contain only the symbols A, X, and 9, but not all A's or all 9's.

Alphanumeric data can be initialized in the Data Division with the VALUE clause. The value specified must be a nonnumeric literal (a character-string enclosed in double quotes) or a figurative constant. Alphanumeric data can be initialized in the Procedure Division with the ACCEPT, MOVE, READ...INTO, and DISPLAY AND READ statements.

The following program segment uses the EXTRACT subroutine to illustrate initialization of alphanumeric data by means of the VALUE clause and the ACCEPT and MOVE statements.

Example:

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
 *THE NEXT LINE ILLUSTRATES INITIALIZATION BY THE VALUE CLAUSE.
  77  CURRENT-LIBRARY-KEYWORD X(2) VALUE ''CL''.
  77  CURRENT-LIBRARY-RECEIVER PIC X(8).
```

```
*THE NEXT ITEM IS INITIALIZED IN THE PROCEDURE DIVISION BY MOVE.
 77  CURRENT-VOLUME-KEYWORD PIC XX.
 77  CURRENT-VOLUME-RECEIVER PIC X(6).
*THE NEXT ITEM IS INITIALIZED IN THE PROCEDURE DIVISION BY ACCEPT.
 77  KEYWORD-3 PIC XX.
 77  RECEIVER-3 PIC X(8).
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
     MOVE ''CV'' TO CURRENT-VOLUME-KEYWORD.
     ACCEPT KEYWORD-3.
     CALL ''EXTRACT'' USING CURRENT-LIBRARY-KEYWORD,
        CURRENT-LIBRARY-RECEIVER, CURRENT-VOLUME-KEYWORD,
        CURRENT-VOLUME-RECEIVER, KEYWORD-3, RECEIVER-3.
```

Integer Data

To define a data item as an integer, you must code the USAGE IS BINARY clause in the data description entry.

COBOL integer items are stored in halfword (two-byte) binary format. The subroutines, however, accept only four-byte integer arguments. You can solve this problem by defining a four-byte group BINARY item composed of two elementary items. For example:

```
01  GROUP-ITEM USAGE BINARY.
    03  FILLER VALUE ZERO.
    03  INTEGER-DATA.
```

The CALL USING statement passes GROUP-ITEM to the subroutine. If you use GROUP-ITEM to send data to the subroutine, initialize FILLER to zero. The subroutine then receives the integer contained in INTEGER-DATA. If you use GROUP-ITEM to receive integer data from the subroutine, the calling program references the elementary item INTEGER-DATA rather than GROUP-ITEM on return from the subroutine.

You must use other methods for negative integers and integers greater than 32767. These methods are explained below, after the discussion of initializing integer data.

Use the VALUE clause to initialize integer data in the Data Division. The value that you specify must be a numeric literal (not enclosed in quotes) containing only digits or a figurative constant ZERO.

There are several methods you can use to initialize integer items in the Procedure Division. The COMPUTE, MOVE, PERFORM...VARYING, and READ...INTO statements initialize integer items directly. You can use the ACCEPT statement to enter character representations of integers; integer data items can then be initialized by converting the character representations to their numeric values. Perform the conversion by using the MOVE or MOVE WITH CONVERSION statement or a BASIC subroutine using the CONVERT statement, as explained later in this subsection. The DISPLAY AND READ statement can initialize integer items by transferring data entered at the workstation to an OBJECT field of the workstation screen description entry.

In the program segment that follows, the READVTOC subroutine returns the names of the files in a library 10 files at a time, beginning with the first file in the library. This segment illustrates initialization by means of the COMPUTE, MOVE, and PERFORM...VARYING statements, but is not meant to illustrate realistic programming practice.

```
 WORKING-STORAGE SECTION.
*THE NEXT ITEM IS INITIALIZED IN THE PROCEDURE DIVISION BY COMPUTE.
 77 COMPUTABLE USAGE BINARY.
 77 TY-PE PIC X VALUE ''F''.
 77 LIB-RARY PIC X(8).
 77 VOL-UME PIC X(6).
 01 STARTER.
*TWO ELEMENTARY BINARY ITEMS FOLLOW.  THE FIRST IS INITIALIZED
*HERE BY THE VALUE CLAUSE.  THE SECOND IS INITIALIZED IN THE
*PROCEDURE DIVISION BY PERFORM VARYING.
    03 FILLER USAGE IS BINARY VALUE ZERO.
    03 STARTNUMBER  USAGE IS BINARY.
 01 COUN-TER.
    03 FILLER USAGE IS BINARY VALUE ZERO.
*THE NEXT ITEM IS INITIALIZED IN THE PROCEDURE DIVISION BY MOVE.
    03 COUNTNUMBER USAGE IS BINARY.
 77 RECEIVER PIC X(80).
 01 RETURNCODE.
    03 FILLER USAGE IS BINARY VALUE ZERO.
    03 RETURNVALUE USAGE IS BINARY.
 01 FILE-COUNT.
    03 FILLER USAGE IS BINARY.
    03 FILECOUNT  USAGE IS BINARY.
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
    ACCEPT LIB-RARY, VOL-UME.
    COMPUTE COMPUTABLE = 10 ** 1.
    MOVE COMPUTABLE TO COUNTNUMBER.
    PERFORM CALL-PARAGRAPH VARYING STARTNUMBER FROM 1 BY 10
      UNTIL COUNTNUMBER LESS THAN 10.
 CALL-PARAGRAPH.
    MOVE SPACES TO RECEIVER.
    CALL ''READVTOC'' USING TY-PE, LIB-RARY, VOL-UME, STARTER,
      COUN-TER, RECEIVER, RETURNCODE, FILE-COUNT.
```

You can use negative integers or integers greater than 32767 by writing BASIC subroutines that use the CONVERT statement. For example, the EXTRACT user subroutine obtains the size of a program's Segment 2 area, which is always greater than 32767. To get the Segment 2 size, the COBOL program must provide EXTRACT with a four-byte numeric receiver. Since the left-most bit of this field is used for the sign, the value received from EXTRACT cannot be interpreted as an integer. The BASIC CONVERT statement, however, can convert the four-byte item to a nine-byte item whose contents represent the sign and the integer value, although not in integer format. The following is an example of the COBOL code necessary to call a BASIC subroutine named 4TO9, which converts data from four bytes to nine bytes.

```
77 KEYWORD PIC X(2) VALUE ''S2''.
01 TEMP PIC X(4).
77 SEG-2-SIZE  PIC S9(8).
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
    CALL ''EXTRACT'' USING KEYWORD, TEMP.
    CALL ''4TO9'' USING TEMP, SEG-2-SIZE.
```

The BASIC subroutine requires two parameters from the COBOL program: a four-byte item and an eight-byte signed item. The BASIC subroutine receives the contents of the four-byte item and converts it to a nine-byte item, with one byte for the sign and eight bytes for the value. The following is a BASIC subroutine that performs the conversion.

```
10 SUB ''4T09'' ADDR (COBOL4%, COBOL9$)
20 DIM COBOL9$9
40 CONVERT COBOL4% TO COBOL9$, PIC(+########)
50 END
```

You can use another BASIC subroutine to convert nine-byte alphanumeric data to four-byte integer data that can be negative or greater than 32767. The integer data can then be passed to a user subroutine. The subroutine follows.

```
10 SUB ''9T04'' ADDR (COBOL9$, COBOL4%)
20 DIM COBOL9$9
30 CONVERT COBOL9$ TO COBOL4%
40 END
```

You can employ a BASIC subroutine like 9T04 to invoke a user subroutine interactively, supplying values for data items by means of the COBOL ACCEPT statement, which transfers only alphanumeric data. The BASIC subroutine converts alphanumeric data to the integer data required by the user subroutine. The following COBOL program segment demonstrates how to use the SET subroutine interactively to change the default lines per page for printer output. The keyword "LI" informs SET that the integer value passed is the number of lines per page.

```
77 LINES-CODE PIC X(2) VALUE ''LI''.
01 LINES-VALUE.
   03 SIGN-ITEM PIC X VALUE ''+''.
   03 LINES-NUM PIC X(8).
01 LINES-PER PIC X(4).
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
    DISPLAY ''TYPE IN LINES-NUM.''.
    ACCEPT LINES-NUM.
    CALL ''9T04'' USING LINES-VALUE, LINES-PER.
    CALL ''SET'' USING LINES-CODE, LINES-PER.
    STOP RUN.
```

Integers from -1 to -32768 can be passed without the use of a BASIC subroutine. First, define a group item composed of two BINARY items, as above. Second, the program moves HIGH-VALUES to the group item, then moves a negative numeric item to the low-order elementary item. In two's-complement notation, the HIGH-VALUES move has the effect of propagating the negative sign across the high-order half of the group item. For a positive number, the program moves LOW-VALUES.

In the following program segment, the G+ function of the DATE subroutine adds a negative number to a given Gregorian date to determine the earlier date.

```
77 FUNCTION  PIC X(2) VALUE ''G+''.
77 START-DATE  PIC X(6) VALUE ''810717''.
77 ADD-DAYS  PIC S9(4) VALUE -0001.
```

```
01 INTEGER-DAYS.
   03 FILLER USAGE IS BINARY VALUE ZERO.
   03 HALFWORD-DAYS USAGE IS BINARY.
77 END-DATE PIC X(6).
01 RETURN-KODE.
   03 FILLER USAGE BINARY VALUE ZERO.
   03 RETURNED USAGE BINARY.
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
    MOVE HIGH-VALUES TO INTEGER-DAYS.
    MOVE ADD-DAYS TO HALFWORD-DAYS.
    CALL ''DATE'' USING FUNCTION, START-DATE, INTEGER-DAYS,
      END-DATE, RETURN-KODE.
    DISPLAY END-DATE.
```

2.2.3 FORTRAN Language

Calling the Subroutine

The form of the FORTRAN language CALL statement is as follows:

CALL subname (arguments)

Subname is the name of the subroutine. Because subroutine names cannot exceed six characters, each subroutine whose name is longer has a note indicating the six-character name that must be used.

Arguments are enclosed within parentheses and are separated by commas. The order in which the arguments appear must be the same as that specified in the argument list. Also, each argument must agree in type and size with the corresponding argument in the list.

Alphanumeric Data

Specify an alphanumeric constant by enclosing its value within single quotes.

Example:

```
OPT = 'FC'
PCLASS = '#'
CALL SET (OPT, PCLASS)

are equivalent to
CALL SET ('FC','#')
```

You can declare variables having alphanumeric values in specification statements (such as LOGICAL, INTEGER, or REAL). A variable having alphanumeric data can be any data type, although the number of characters it requires might determine which type is most appropriate. Table 2-1 provides examples of variable sizes and specification statements that define the space required by the variable ("name" indicates the variable name).

Table 2-1. Alphanumeric Size and FORTRAN Specification Statements

Number of Characters	Specification Statement
1	LOGICAL*1 name
2	INTEGER*2 name
3	LOGICAL*1 name(3)
4	INTEGER name or LOGICAL name
6	LOGICAL*1 name(6)
8	REAL*8 name
10	LOGICAL*1 name(10)
16	REAL*8 name(2)
22	LOGICAL*1 name(22)

Integer Data

The program specifies integer data by indicating its value.

Example:

```
NSECS = 10
CALL BELL (NSECS)

are equivalent to
CALL BELL (10)
```

Designate a variable as integer data type by beginning its name with a letter between I and N, or by including its name in an INTEGER or IMPLICIT specification statement. The following statements illustrate how to declare a variable (PRINTR) as integer type.

Example:

```
INTEGER PRINTR
NFORM = 0
PRINTR = 10
CALL SET ('FN',NFORM,'PR',PRINTR)
```

Integer variables and constants are stored in four bytes by default.

Use of Files with the Subroutines

Some subroutines permit the use of files for output and require that the pointer to the file UFB be identified so that the necessary file information is present. FORTRAN does not provide the pointer to the UFB. To use a subroutine that requires a UFB address, you must code the call to the USERSUBS subroutine as either a BASIC or COBOL subroutine and link that subroutine to the program. The appropriate programming language reference manual provides additional information.

2.2.4 PL/I Language

Declaring the Subroutine

A PL/I subroutine accesses the user subroutines as external procedures. PL/I programs must declare the names of these procedures with the ENTRY attribute. The ENTRY declaration must also indicate the data types of all arguments passed to or from the subroutine. Because the user subroutines pass only alphanumeric and four-byte integer data, the ENTRY declaration should specify only CHARACTER and FIXED BINARY(31) data types. For example, a PL/I program that calls the SET subroutine must contain the following declaration:

```
DECLARE SET ENTRY (CHARACTER(2), CHARACTER(1));
```

Calling the Subroutine

The form of the PL/I CALL statement is as follows:

```
CALL subname (arguments);
```

Subname is the name of the subroutine. It must have been previously declared with the ENTRY attribute. Arguments must be enclosed within parentheses and separated by commas. The arguments must appear in the order specified in the argument list. To prevent undesirable data type conversion, each argument must agree in type and size with the corresponding argument in the list.

Alphanumeric Data

Variables with alphanumeric values should be declared with the CHARACTER data type and the length of the character string. Variables with the STATIC storage class can be assigned initial values in the declaration statement. Alphanumeric constants are specified by enclosing their values within single or double quotes.

Example:

```
DECLARE OPT CHARACTER(2), PCLASS CHARACTER(1);
OPT = 'FC';
PCLASS = '#';
CALL SET (OPT, PCLASS);

are equivalent to
DECLARE OPT STATIC CHARACTER(2) INIT(''FC'');
DECLARE PCLASS STATIC CHARACTER(1) INIT('#');
CALL SET (OPT, PCLASS);

are equivalent to
CALL SET ('FC',''#'');
```

Integer Data

Variables with integer values should be declared with the FIXED BINARY(31) data type. Variables with the STATIC storage class can be assigned initial values in the declaration statement. Integer constants are specified by indicating an integer value (i.e., 4). Because integer constants are assigned the FIXED DECIMAL data type by the PL/I

compiler, integer constants are automatically converted to the FIXED BINARY data type by the PL/I compiler when passed to the user subroutine.

2.2.5 RPG II Language

Calling the Subroutine

To call USERSUBS subroutines, RPG II programs use the User Aid RPGCALL. RPGCALL creates an interface between the calling RPG II program and the USERSUBS subroutine so that arguments can be passed back and forth. To access RPGCALL, you must write a one-statement Assembly language program. This subsection explains how to code and use that program in calling USERSUBS subroutines. For additional information about calling subroutines in RPG II, refer to the *VS RPG II Language Reference*.

In RPG II, the EXIT operation code indicates the point at which flow of control passes from a calling program to a subroutine. Factor 2 specifies the name of the subroutine that is to receive control; factor 1, the result field, and the resulting indicators must be left blank.

Use the RLABL operation code to pass arguments from the calling program to the subroutine. Each argument that is passed requires one RLABL statement; name the argument in the result field. Factor 1, factor 2, conditioning indicator (columns 9-17), and resulting indicator entries must be left blank. The RLABL statements for a subroutine call can appear anywhere in the calculations. The ULABL operation code is not used in calling USERSUBS subroutines. After execution of the subroutine, control returns to the first executable statement after the EXIT statement.

You must take the following steps when using RPGCALL to call a USERSUBS subroutine:

Step 1. Be sure that RPGCALL is stored on the system. RPGCALL is available from the International Society of Wang Users (ISWU), Wang Laboratories, Inc., One Industrial Avenue, Lowell, MA 01851, Tel. (617) 459-5000.

Step 2. Write and assemble the short Assembler program described below. At assemble time, supply the name of the library on which the RPGCALL program resides.

Step 3. Write and compile the calling program. In the EXIT statement, include the name of the assembled program file from Step 2 (instead of the name of the USERSUBS subroutine). The RLABL statements must list the arguments that are to be passed to the USERSUBS subroutine. (The arguments are passed to the Assembler program, which then passes them to the USERSUBS subroutine.)

Step 4. Run the LINKER, either directly from the Command Processor or as an option when compiling the calling program from the EDITOR. You must link three program files: the calling program, the USERSUBS subroutine, and the assembled program file from Step 2. The result of the LINKER's execution is one executable program file.

Format the short Assembler program as follows:

```
         1         2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
         RPGCALL   NAME=xxxxxx,CALL=yyyyyy,(arguments)                  C
               (arguments-continued)
```

The word RPGCALL begins in column 10 of the first line. The remainder of the statement starts in column 20 with the word NAME. If all the arguments cannot fit on one line, a C in column 72 denotes a continuation. All continuation lines begin in column 16.

The fields have the following meanings:

xxxxxx — The name of the assembled version of the one-statement Assembler program. This name is included in factor 2 of the EXIT statement in the calling RPG II program. It cannot be longer than six characters and must be unique (not used for any other purpose in the program).

yyyyyy — The name of the USERSUBS subroutine that the program is calling.

arguments — The list of arguments to be passed between the USERSUBS subroutine and the calling program. The arguments must be in the order and of the type expected by the subroutine.

This Assembler statement tells the RPGCALL macro which subroutine is being called and which arguments are being passed. The calling program specifies the arguments in RLABL statements, as described earlier; the EXIT statement is used to transfer control to the one-statement Assembler program. When the EXIT statement is executed, RPGCALL calls the USERSUBS subroutine, resolving memory addresses, and sometimes converting data types.

Each parameter in the list can be expressed in either of the following formats:

FORMAT A:

```
FIELD
```

This format is used to pass an alphanumeric field.

FORMAT B:

```
(FIELD, DIGITS, F)
```

This format is used to pass an integer field. RPG II programs store all numeric fields internally in packed decimal format, while the USERSUBS subroutines require integer data in fullword binary format. RPGCALL performs the necessary conversions.

Alphanumeric Data and Variables

An alphanumeric field can have any valid RPG II field name: six characters (numerals and letters only), beginning with a letter. Field names cannot contain embedded blanks. A blank decimal position entry defines a field as alphanumeric. Alphanumeric literals can contain any ASCII characters and are always enclosed in quotes.

Integer Data and Variables

An integer field can have any valid RPG II field name, as described above. A decimal position entry of 0 defines a field as integer. Integer constants are not enclosed in quotes.

Use of Files with the Subroutines

Some subroutines permit the use of files for output and require that the pointer to the file UFB be identified so that the necessary file information is present. RPG II does not provide the pointer to the UFB. To use a subroutine that requires a UFB address, you must code the call to the USERSUBS subroutine as either a BASIC or COBOL subroutine and link that subroutine to the program. The appropriate programming language reference manual provides additional information.

2.2.6 How to Link Subroutines with Programs

To use these subroutines, you must link the program with the USERSUBS subroutine. There are two ways to perform this link:

1. Through the EDITOR
2. Through the LINKER

Each method is described below. More detailed information on editing and linking appears in the *VS Program Development Tools*.

Linking through the EDITOR

You can use the EDITOR to link a subroutine with a program that is being compiled. From the EDITOR special menu, PF9 (RUN) compiles and runs the program. Make the following changes to the Linker screen: LINK=YES causes linking to occur, and LIBRARY=USERSUBS searches that library for references to subroutines not contained in the program. Note that the library name should correspond to the library in which these subroutines reside on your system.

If you are a FORTRAN programmer linking individual files by using the Linkfile screen instead of specifying the subroutine library name, you must add a step when accessing user subroutines whose usual names exceed six characters (e.g., GETPARM, EXTRACT). In this case, you must provide the name and location of both the shortened name (e.g., GETPRM, XTRACT) and the full name on the Linkfile screen.

If you are programming in RPG II, you must also link the short Assembly language program that calls the RPGCALL macro. (Refer to Section 2.2.5.)

Linking with the LINKER

The LINKER combines a number of separately compiled program units to form a single executable program file. Use the LINKER to link your program with a USERSUBS subroutine when you have already compiled your program and saved the program file.

Using the LINKER involves the following steps. First, invoke the LINKER by pressing PF1 (RUN) from the Command Processor. Enter LINKER as the program file. Then, on

the Options screen, specify the library that contains the USERSUBS subroutine to be linked with your program. Next, specify your program as an input file on an Input screen. (It is not necessary to specify more than one input file.) Finally, specify a file name for the program file output on the Output screen. This file contains the compiled program and subroutine. The result is an executable program that can be run directly from the Command Processor.

If you are a FORTRAN programmer linking individual files by using Input screens instead of specifying the subroutine library name, you must add a step when accessing user subroutines whose usual names exceed 6 characters (e.g., GETPARM, EXTRACT). In this case, you must specify the usual subroutine name and the shortened name on separate Input screens. For example, to access EXTRACT, you must link EXTRACT *and* XTRACT by specifying those names on separate Input screens. You may, of course, just specify the subroutine library on the Options or Library screen and not add this extra step.

RPG II programmers must also link the short Assembly language program that calls the RPGCALL macro. (Refer to Section 2.2.5)

When a subroutine is revised, making it necessary to replace its program file, the LINKER can make this replacement. Refer to the *VS Program Development Tools* for more information. Note that it may also be necessary to revise the calling sequence and recompile the program.

How to Find the Subroutine Version Number

You can obtain the version number of a USERSUBS subroutine by running the DISPLAY utility and displaying the subroutine's object code. The object code appears as a sequence of random characters. The subroutine version number appears near the beginning of the code.

CHAPTER 3
SUBROUTINE DESCRIPTIONS

BELL

FUNCTION

Sounds the workstation alarm for a user-specified amount of time.

USAGE (arg1)

Pos	Argument	Type	Size	Comments
arg1	Time	Integer	4	Amount of time to sound the workstation alarm, in tenths of a second. If zero or negative, the alarm is not sounded.

NOTE

The workstation must be closed before the program calls this subroutine (the calling statement cannot be immediately preceded by any statement that accesses the workstation, either for input or for output). In BASIC, the CLOSE WS statement closes the workstation.

BELL Subroutine — A FORTRAN Example

This program causes the workstation alarm to sound for 3/10 of a second.

```
C  SOUND THE WORKSTATION ALARM FOR 3/10 SECOND
      ITIME = 3
C  CALL BELL SUBROUTINE WITH 'ITIME' ARGUMENT
      CALL BELL (ITIME)
      END
```

BITPACK

FUNCTION

Converts a binary string into its ASCII character equivalent.

USAGE (arg1, ..., arg3)

Pos	Argument	Type	Size	Comments
arg1	Binary String	Alpha	var	Binary string to convert, supplied by user program. Length must be a multiple of 8.
arg2	Receiver	Alpha	var	ASCII equivalent of the input string, returned by the subroutine. Must be at least 1/8th the length of the input string.
arg3	Length	Integer	4	Length of the input string; must be a multiple of eight (any excess digits are ignored).

NOTES

1. The subroutine does not check to ensure that the input string is binary.
2. For FORTRAN programs, the name of this subroutine must be specified as BTPACK.

BITPACK Subroutine — A FORTRAN Example

This program requests that the user input a binary number from the workstation. The program then converts the number to its ASCII equivalent and displays it on the workstation.

```
C   'RCVR' IS THE 1-CHARACTER ASCII EQUIVALENT TO THE BINARY STRING
        LOGICAL*1 RCVR
C   'STRING' IS AN 8-CHARACTER BINARY NUMBER
        REAL*8 STRING
        WRITE(0,101) ' ENTER 8 BINARY DIGITS:'
        READ(0,102) STRING
C    END PROGRAM IF STRING = 11111111
        IF(STRING .EQ. '11111111') GO TO 99
C
C    CALL BITPACK SUBROUTINE ('BTPACK' IN FORTRAN)
        CALL BTPACK(STRING,RCVR,8)
C
        WRITE(0,103) RCVR
  101   FORMAT(A23)
  102   FORMAT(A8)
  103   FORMAT(1X,'ASCII: ',A1)
   99   PAUSE
        END
```

BITUNPK

FUNCTION

Converts an ASCII character string into its binary equivalent.

USAGE (arg1, ..., arg3)

Pos	Argument	Type	Size	Comments
arg1	ASCII String	Alpha	var	String of ASCII characters to be converted, supplied by the user program.
arg2	Receiver	Alpha	var	Binary string, returned by the subroutine. The length of the receiver must be at least 8 times the length of the input string.
arg3	Length	Integer	4	Length of the input string.

NOTE

For FORTRAN programs, the name of this subroutine must be specified as BTUNPK.

BITUNPK Subroutine — A FORTRAN Example

This example requests that the user input an ASCII string from the workstation. The program then converts the string to its binary equivalent and displays it on the workstation.

```
C  'OUT' CAN HOLD UP TO 24 SEPARATE CHARACTERS
       REAL*8 OUT(3)
       WRITE(0,101)
       READ(0,102) IN
C  USER ENTERS 'QQQ' TO STOP
       IF(IN .EQ. 3HQQQ) GO TO 99
C
C  CALL BITUNPK ('BTUNPK' IN FORTRAN)
       CALL BTUNPK (IN, OUT, 3)
C
       WRITE(0,103) OUT
  101  FORMAT(' ENTER 1-3 CHARACTERS (QQQ TO STOP)')
  102  FORMAT(A3)
  103  FORMAT(' BINARY:',3A8)
   99  PAUSE
       END
```

CANCEL

FUNCTION

Cancels execution of the calling program and displays a message on the workstation. The message consists of a message ID, a message issuer, and a message that can be several lines in length.

USAGE (arg1, ..., arg5)

Pos	Argument	Type	Size	Comments
arg1	Msg ID	Alpha	4	Message identification, supplied by the user program.
arg2	Message Issuer	Alpha	6	Message issuer identifier, supplied by the user program.
arg3	Msg Text Line Count	Integer	4	Number of message text lines. The program can specify the message as separate text lines (include arg3), or as a block containing the complete text (omit arg3). If arg3 is specified, arg4 and arg5 are repeated for each text line. If arg3 is omitted, see arg4 for action.
arg4	Message Text	Alpha	var	Message to be displayed. *Arg3 specified:* arg4 is a single line of text, containing no embedded X'0D' characters. Each line can begin with the following control characters, singly or in combination: X'5E' (up-arrow) — center msg text X'5F' (underscore) — underline msg text X'21' (exclamation pt) — blink msg text *Arg3 omitted:* the message can consist of several lines of text, where lines are separated by a single X'0D' character. No control characters are recognized.
arg5	Msg Text Length	Integer	4	Length of message text. Include control characters in text length. A text length of zero (excluding control characters) generates no text line. If the argument list consists only of empty text strings, the subroutine generates a single blank as the message. *Arg3 specified:* length of text line (arg4). *Arg3 omitted:* length of entire msg (arg4).

NOTE

CANCEL terminates the program, displays a message on the workstation, and allows the user to enter debug processing or cancel processing.

CANCEL Subroutine — A BASIC Example

This program terminates execution and displays a message on the screen. The user supplies the message ID, issuer, and the cancel message.

```
000100DIM MESSAGEID$4,ISSUER$6,MESSAGE$60
000200ACCEPT                                                                  !
000300       AT (01,25),                                                      !
000400''DEMONSTRATION OF CANCEL SUBROUTINE'',                                 !
000500       AT (08,03),                                                      !
000600''Message ID:'',                                                        !
000700       AT (08,20), MESSAGEID$    , CH(04),                            !
000800       AT (09,03),                                                      !
000900''Issuer:'',                                                            !
001000       AT (09,20), ISSUER$       , CH(06),                            !
001100       AT (10,03),                                                      !
001200''Cancel Message:'',                                                    !
001300       AT (10,20), MESSAGE$      , CH(60),                            !
001400       AT (14,03),                                                      !
001500''Fill in the information and press ENTER.  The program will cance!
0016001 with the'',                                                           !
001700       AT (15,03),                                                      !
001800''above information.''
001900    CALL ''CANCEL''  ADDR(MESSAGEID$,ISSUER$,MESSAGE$,60%)
```

CEXIT

FUNCTION

Overrides system cancel processing.

On abnormal program termination, the user can press PF1 to enter debug processing, PF16 to cancel processing, or the HELP key to access the Modified Command Processor. CEXIT allows the programmer to restrict debug processing, to associate PF16 with alternate processing, and to disable operation of the HELP key.

USAGE (arg1, ..., arg5)

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Indicates whether to set or cancel options: S = Set options. C = Cancel options (no further arguments are required).
arg2	Cancel Option	Alpha	1	Indicates whether to allow debug processing after abnormal program termination: Blank = Normal cancel processing (default). N = No debug processing. D = No debug processing. Provide dump. Optional. It might not be desirable to initiate debug processing after abnormal program termination when the user is not the program developer.
arg3	HELP Key Option	Alpha	1	Action of HELP key: H = Enable HELP key (default). N = Disable HELP key. Disabling the HELP key might be desirable when the program operator should not have access to the Command Processor.
arg4	PF16 Message	Alpha	var	Allows replacement of the PF16 message for cancel option after abnormal termination. Default is no message replacement. If included, arg5 must be included.
arg5	PF16 Msg Length	Integer	4	Length of PF16 message. Maximum of 27 characters. Must be included if arg4 is present.

NOTE

Arguments 2 through 5 are optional. However, if any are included, all preceding arguments must be included.

CEXIT Subroutine — A COBOL Example

This program sets the Nodebug option and disables the HELP key for a cancel exit.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CEXITC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600*THE FOLLOWING ARE THE FIRST THREE ITEMS FROM THE CEXIT ARGUMENT
000700*LIST.
000800 77  CEXIT-TYPE  PIC X VALUE ''S''.
000900 77  C-OPTION  PIC X VALUE ''N''.
001000 77  HELP-OPTION  PIC X VALUE ''N''.
001100 PROCEDURE DIVISION.
001200 MAIN-PARAGRAPH.
001300     CALL ''CEXIT'' USING CEXIT-TYPE, C-OPTION, HELP-OPTION.
001400*THE NEXT INSTRUCTION ALLOWS THE USER TO TEST THE RESULTS OF THE
001500*DISABLED HELP KEY OPTION BY PRESSING THE HELP KEY WHILE THE
001600*SCREEN IS DISPLAYED.
001700     DISPLAY ''THE HELP KEY IS DISABLED.''.
001800     STOP RUN.
```

CHKPARM

FUNCTION

Performs table checking on one or more data fields entered previously by a user or procedure. It can be used for any type of field checking but is primarily intended for GETPARM Limited Alphanumeric and Alphanumeric keyword field types (refer to the GETPARM subroutine). CHKPARM can optionally identify abbreviations of various lengths for the table entries it is checking.

USAGE (arg1, ..., arg6) for each keyword to be checked

The CHKPARM subroutine argument list consists of one or more sets of arguments, each consisting of six arguments. There is one set for each GETPARM keyword field that the subroutine checks.

Pos	Argument	Type	Size	Comments
arg1	Field	Alpha	var	Name of data field whose value is to be checked.
arg2	Length	Integer	4	Field length. It must be positive and cannot exceed 256.
arg3	Table Size	Integer	4	Number of comparison strings in the table that follows, against which the subroutine checks data values. It must be positive.
arg4	String Table	Alpha	var	Character string array, which is the table of comparison strings. The length of each table element must be that of the keyword field itself, as specified in arg2. The table check proceeds in element order, starting from the first element, until either a match is found or the table is exhausted.
arg5	Length Table/Flag	See Note	1	See Note 1 for information.
arg6	Ret. Code	Integer	4	Return code, set to the table element number that matches the keyword field. If the subroutine does not find a match, the return code is set to zero.

NOTES

1. The program can use argument 5 to indicate legal abbreviations for the acceptable field values (e.g., "Y" or "YE" allowed for "YES"); these abbreviations can be either a letter (N or A) or an integer table. If no such abbreviations are to be allowed, then the program specifies N for this argument. Conversely, if all possible abbreviations (which must be at least one character) are to be allowed, the program specifies A (for all abbreviations).

   For special cases in which some, but not all, abbreviations are to be allowed, neither N nor A is adequate. This argument becomes, instead, a table of "minimum lengths." This table is in the form of an integer array having exactly as many elements as the compare string table (arg4), with each integer element corresponding to the comparably placed string element. The integer value is the minimum number of compare string characters that must be present in the keyword field in order to recognize a match. For example, a compare string of "INDEX" and a minimum length of 3 matches keyword fields "IND ", "INDE ", and "INDEX", but will not match "IN ", since it has fewer than 3 of the compare string characters. A minimum length of 0 matches any abbreviation of the compare string, and also matches a completely blank field (used for "default" values); a minimum length that is equal to the field length (arg2) has the same effect as "no abbreviation"; a minimum length greater than the field length specifies "never match." Finally, a minimum length table containing all 1 values has the same effect as specifying argument value A, rather than passing the entire table (see above).

2. For FORTRAN programs, the name of this subroutine must be specified as CHKPRM.

CHKPARM Subroutine — A COBOL Example

This program calls the GETPARM subroutine to solicit parameters for an output file. It then calls the CHKPARM subroutine to check which of four possible values was entered in response to the GETPARM request for the file's device type. The program instructs CHKPARM to accept abbreviations for the device types. Each device type has a different length abbreviation.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CHKPARMC.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500 FIGURATIVE-CONSTANTS.
000600     CENTER IS ''5E''.
000700     BLINK IS ''21''.
000800 DATA DIVISION.
000900 WORKING-STORAGE SECTION.
001000*THE FOLLOWING ITEMS ARE PARAMETERS FOR THE GETPARM SUBROUTINE
001100 77  TY-PE PIC X(2) VALUE ''I''.
001200 77  FO-RM PIC X VALUE ''R''.
001300 77  PR-NAME PIC X(8) VALUE ''OUTPUT''.
001400 77  KEY-RECEIVER PIC X(1).
001500 77  MESSAGE-NUMBER  PIC X(4) VALUE ''9999''.
001600 77  MESS-ENGER PIC X(7) VALUE ''CHKPARM''.
001700*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001800*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001900*HALFWORD-BINARY ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
002000*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
002100*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
002200 01  LINE-COUNT.
002300     03 FILLER USAGE IS BINARY VALUE 0.
002400     03 LINE-OFFSET USAGE IS BINARY VALUE 1.
002500 01  MESS-AGE.
002600     03 CONTROL-1 PIC X VALUE CENTER.
002700     03 CONTROL-2 PIC X VALUE BLINK.
002800     03 TEXT PIC X(27) VALUE ''PLEASE SUPPLY THESE VALUES''.
002900 01  MESSAGE-LENGTH.
003000     03 FILLER USAGE IS BINARY VALUE 0.
003100     03 M-LENGTH USAGE IS BINARY VALUE 29.
003200 77  KEYWORD-TYPE PIC X VALUE ''K''.
003300 77  KEYWORD-1 PIC X(8) VALUE ''FILE''.
003400 77  VALUE-1 PIC X(8) VALUE SPACES.
003500 01  VALUE-LENGTH.
003600     03 FILLER USAGE BINARY VALUE 0.
003700     03 LENGTH USAGE BINARY VALUE 8.
003800 01  ROW-1.
003900     03 FILLER USAGE IS BINARY VALUE 0.
004000     03 ROW-VALUE-1 USAGE IS BINARY VALUE 1.
004100 01  COLUMN-1.
004200     03 FILLER USAGE IS BINARY VALUE 0.
004300     03 COLUMN-VALUE-1 USAGE IS BINARY VALUE 10.
```

```
004400 77  DATA-TYPE PIC X(2) VALUE ''L''.
004500 77  KEYWORD-2 PIC X(8) VALUE ''LIBRARY''.
004600 77  VALUE-2 PIC X(8) VALUE SPACES.
004700 01  ROW-2.
004800      03 FILLER USAGE IS BINARY VALUE 0.
004900      03 ROW-VALUE-2 USAGE IS BINARY VALUE 4.
005000 77  KEYWORD-3 PIC X(6) VALUE ''VOLUME''.
005100 77  VALUE-3 PIC X(6) VALUE SPACES.
005200 01  VALUE-3-LENGTH.
005300      03 FILLER USAGE IS BINARY VALUE 0.
005400      03 VOLUME-LENGTH USAGE IS BINARY VALUE 6.
005500 01  ROW-3.
005600      03 FILLER USAGE IS BINARY VALUE 0.
005700      03 ROW-VALUE-3 USAGE IS BINARY VALUE 4.
005800 77  KEYWORD-4 PIC X(6) VALUE ''DEVICE''.
005900*THE FOLLOWING IS THE GETPARM ITEM THAT WILL BE CHECKED BY CHKPARM
006000 77  VALUE-4 PIC X(7) VALUE SPACES.
006100*THE NEXT ITEM IS PASSED BOTH TO GETPARM AND CHKPARM.
006200 01  VALUE-4-LENGTH.
006300      03 FILLER USAGE IS BINARY VALUE 0.
006400      03 DEVICE-LENGTH USAGE IS BINARY VALUE 7.
006500 01  ROW-4.
006600      03 FILLER USAGE IS BINARY VALUE 0.
006700      03 ROW-VALUE-4 USAGE IS BINARY VALUE 4.
006800*THE NEXT ITEM CONTAINS THE VALUES TO BE CHECKED BY CHKPARM
006900 01  DEVICES.
007000      03 FILLER PIC X(7) VALUE ''DISK''.
007100      03 FILLER PIC X(7) VALUE ''DISPLAY''.
007200      03 FILLER PIC X(7) VALUE ''PRINTER''.
007300      03 FILLER PIC X(7) VALUE ''TAPE''.
007400 01  DEVICE-TABLE REDEFINES DEVICES.
007500      03 DEVICE PIC X(7) OCCURS 4 TIMES.
007600 01  DEVICE-TABLE-SIZE.
007700      03 FILLER USAGE IS BINARY VALUE 0.
007800      03 DEVICE-TABLE-LENGTH USAGE BINARY VALUE 4.
007900 01  LENGTHS.
008000      03 INTEGER-1.
008100          05 FILLER USAGE BINARY VALUE 0.
008200          05 LENGTH-1 USAGE BINARY VALUE 3.
008300      03 INTEGER-2.
008400          05 FILLER USAGE BINARY VALUE 0.
008500          05 LENGTH-2 USAGE BINARY VALUE 4.
008600      03 INTEGER-3.
008700          05 FILLER USAGE BINARY VALUE 0.
008800          05 LENGTH-3 USAGE BINARY VALUE 5.
008900      03 INTEGER-4.
009000          05 FILLER USAGE BINARY VALUE 0.
009100          05 LENGTH-4 USAGE BINARY VALUE 2.
009200 01  LENGTH-TABLE REDEFINES LENGTHS.
009300      03 LENGTH-INTEGER OCCURS 4 TIMES.
009400          05 FILLER USAGE BINARY.
009500          05 LENGTH-VALUE USAGE BINARY.
```

```
009600 01  RETURN-KODE.
009700     03 FILLER USAGE BINARY VALUE ZERO.
009800     03 TABLE-ITEM USAGE BINARY.
009900 PROCEDURE DIVISION.
010000 MAIN-PARAGRAPH.
010100     CALL ''GETPARM'' USING TY-PE, FO-RM, PR-NAME, KEY-RECEIVER,
010200         MESSAGE-NUMBER, MESS-ENGER, LINE-COUNT, MESS-AGE,
010300         MESSAGE-LENGTH, KEYWORD-TYPE, KEYWORD-1, VALUE-1,
010400         VALUE-LENGTH, ROW-1, COLUMN-1, DATA-TYPE,
010500         KEYWORD-TYPE, KEYWORD-2, VALUE-2, VALUE-LENGTH, ROW-2,
010600         COLUMN-1, DATA-TYPE, KEYWORD-TYPE,
010700         KEYWORD-3, VALUE-3, VALUE-3-LENGTH, ROW-3,
010800         COLUMN-1, DATA-TYPE,
010900         KEYWORD-TYPE, KEYWORD-4, VALUE-4, VALUE-4-LENGTH,
011000         ROW-3, COLUMN-1, DATA-TYPE.
011100     IF VALUE-1 = ''Z'' STOP RUN.
011200     CALL ''CHKPARM'' USING VALUE-4, VALUE-4-LENGTH,
011300         DEVICE-TABLE-SIZE, DEVICE-TABLE, LENGTH-TABLE,
011400         RETURN-KODE.
011500     DISPLAY TABLE-ITEM.
011600     GO TO MAIN-PARAGRAPH.
```

COMPRESS

FUNCTION

Converts a character string to compressed format. Compressed format can reduce storage for records with repeated characters.

USAGE (arg1, ..., arg5)

Pos	Argument	Type	Size	Comments
arg1	Input	Alpha	var	Character string to be compressed.
arg2	Input length	Integer	4	Length of input string. Must be nonnegative and not greater than 2048.
arg3	Output	Alpha	var	Receiver for compressed string.
arg4	Output Length	Integer	4	Maximum length of output receiver. Must be between 0 and 2048 and is reduced by the subroutine to reflect the actual size of the compressed string.
arg5	Ret. Code	Integer	4	Error return code: 0 =Successful. 4 =Maximum output length (arg4) too short, contents of the output string are unpredictable.

NOTES

1. The operation of this subroutine is identical to the process used by the COMP Assembler instruction, which is used by DMS to generate compressed records.

2. For FORTRAN programs, the name of this subroutine must be specified as CMPRES.

3. This subroutine does the reverse of the EXPAND subroutine.

COMPRESS And EXPAND Subroutines — A COBOL Example

This program calls COMPRESS to compress a character string and displays the compressed string in ASCII characters. It calls HEXUNPK to display the compressed string in hexadecimal characters, calls EXPAND to expand the string, and displays the expanded string.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. COMPRESC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  INPUT-STRING PIC X(21) VALUE ''ABBCCCDDDDEEEEEFFFFFF''.
000700*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000800*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000900*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001000*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001100*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001200 01  INPUT-LENGTH.
001300     03 FILLER USAGE IS BINARY VALUE 0.
001400     03 IN-LENGTH  USAGE IS BINARY VALUE 21.
001500 77  OUTPUT-STRING PIC X(12).
001600 01  OUTPUT-LENGTH.
001700     03 FILLER USAGE IS BINARY VALUE 0.
001800     03 OUT-LENGTH  USAGE IS BINARY VALUE 12.
001900 77  HEX-STRING PIC X(24).
002000 77  EXPANDED-STRING PIC X(21).
002100 01  RETURNCODE.
002200     03 FILLER USAGE IS BINARY VALUE ZERO.
002300     03 ERROR-CODE  USAGE IS BINARY VALUE 0.
002400 PROCEDURE DIVISION.
002500 MAIN-PARAGRAPH.
002600         CALL ''COMPRESS'' USING INPUT-STRING, INPUT-LENGTH,
002700         OUTPUT-STRING, OUTPUT-LENGTH, RETURNCODE.
002800         IF ERROR-CODE NOT = 0, DISPLAY ''OUTPUT LENGTH TOO SHORT'',
002900            GO TO EXIT-PARAGRAPH.
003000         DISPLAY OUTPUT-STRING.
003100         CALL ''HEXUNPK'' USING OUTPUT-STRING, HEX-STRING,
003200            OUTPUT-LENGTH.
003300         DISPLAY HEX-STRING.
003400         CALL ''EXPAND'' USING OUTPUT-STRING, OUTPUT-LENGTH,
003500            EXPANDED-STRING, INPUT-LENGTH, RETURNCODE.
003600         IF ERROR-CODE NOT = 0, DISPLAY ''ERROR CODE = ''ERROR-CODE,
003700            GO TO EXIT-PARAGRAPH.
003800         DISPLAY EXPANDED-STRING.
003900 EXIT-PARAGRAPH.
004000         STOP RUN.
```

DATE

FUNCTION

DATE has several functions that involve the current system date, as well as user-specified dates:

1. Converts current system date and time to a formatted string, suitable for report headings, in uppercase or upper and lowercase.
2. Converts dates between Gregorian and Julian formats (see definitions in USAGE section).
3. Performs calculations with dates, including finding the difference between two dates and obtaining a new date by adding a number of days to a given date.
4. Determines the day of the week corresponding to a given date in the 20th century.

USAGE (arg1, arguments)

Arg1 defines the function and determines the number and nature of the additional arguments.

Several of this subroutine's functions use Gregorian and Julian formats. For example, for the calendar day January 20, 1981, the Gregorian equivalent (in YYMMDD format) is 810120; the Julian equivalent (in YYDDD format, where DDD is the number of days from January 1) is 81020.

1. Get current date and time (uppercase)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is HD
arg2	Date/Time	Alpha	45	Returned by the subroutine, in the following format:

```
AAAAAAAAA   BBBBBBBBBBBBBBBBB   CCCCCCCC
   FRIDAY   JANUARY 20, 1979     2:30 PM
```

2. Get current date and time (upper and lowercase)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is HL
arg2	Date/Time	Alpha	45	Returned by the subroutine, in the following format:

```
AAAAAAAAA   BBBBBBBBBBBBBBBBB   CCCCCCCC
   Friday   January 20, 1979     2:30 PM
```

3. Convert date in Gregorian format to Julian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is GJ
arg2	Greg. Date	Alpha	6	Supplied by user program.
arg3	Jul. Date	Alpha	5	Returned by subroutine.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

4. Convert date in Julian format to Gregorian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is JG
arg2	Jul. Date	Alpha	5	Supplied by user program.
arg3	Greg. Date	Alpha	6	Returned by subroutine.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

5. Compute the difference between two dates in Gregorian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is G-
arg2	Start Date	Alpha	6	Supplied by user program.
arg3	End Date	Alpha	6	Supplied by user program.
arg4	Difference in Days	Integer	4	Returned by subroutine. This value can be positive or negative.
arg5	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

6. Compute the difference between two dates in Julian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is J-
arg2	Start Date	Alpha	5	Supplied by user program.
arg3	End Date	Alpha	5	Supplied by user program.
arg4	Difference in Days	Integer	4	Returned by subroutine. This value can be positive or negative.
arg5	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

7. Add a specified number of days to a Gregorian date to produce a new date

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is G+
arg2	Start Date	Alpha	6	Supplied by user program.
arg3	Days to Add	Integer	4	Supplied by user program. Must be in the range of -36524 to +36525. If outside that range, a return code of 8 results.
arg4	New Date	Alpha	6	Returned by subroutine. If the new date is in the 19th or 21st century, a return code of 4 results.
arg5	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

8. Add a specified number of days to a Julian date to produce a new date

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is J+
arg2	Start Date	Alpha	5	Supplied by user program.
arg3	Days to Add	Integer	4	Supplied by user program. Must be in the range of −36524 to +36525. If outside that range, a return code of 8 results.
arg4	New Date	Alpha	5	Returned by subroutine. If the new date is in the 19th or 21st century, a return code of 4 results.
arg5	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

9. Determine the day of the week from a date in Gregorian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is GD
arg2	Date	Alpha	6	Supplied by user program.
arg3	Day of Week	Alpha	9	Returned by subroutine. The day is uppercase and left-justified.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

10. Determine the day of the week from a date in Julian format

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is JD
arg2	Date	Alpha	5	Supplied by user program.
arg3	Day of Week	Alpha	9	Returned by subroutine. The day is uppercase and left-justified.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-1 below.

NOTE

The subroutine assumes that all dates provided by the user program are in the 20th century. If the subroutine computes a date that is not in the 20th century, a return code of 4 results. If the program then uses that date as an input argument to a subsequent call to the subroutine, DATE assumes that the date is in the 20th century.

Table 3-1. DATE Error Return Codes

Return Code	Meaning
0	Successful operation.
4	The result (for G+ and J+ only) is a year in either the 19th (1800-1899) or 21st century (2000-2099).
8	Invalid input value or format.

DATE Subroutine — A COBOL Example

This program returns the date one day before a specified Gregorian date by adding -1 to the specified date. Since COBOL cannot accept negative integer data, the program uses the method explained in Section 2.2.2 for passing small negative integers.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DATEC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION  PIC X(2) VALUE ''G+''.
000700*THE NEXT ITEM IS THE INPUT DATE.  IT IS INITIALIZED IN THE
000800*PROCEDURE DIVISION.
000900 77  START-DATE  PIC X(6).
001000 77  ADD-DAYS  PIC S9(4) VALUE -0001.
001100*IN THE PROCEDURE DIVISION, ADD-DAYS IS MOVED TO THE LOW-ORDER
001200*TWO-BYTES OF THE FOLLOWING ITEM IN ORDER TO BE PASSED TO THE
001300*SUBROUTINE.
001400 01  INTEGER-DAYS.
001500     03 FILLER USAGE IS BINARY VALUE ZERO.
001600     03 HALFWORD-DAYS USAGE IS BINARY.
001700 77  END-DATE PIC X(6).
001800 01  RETURN-KODE.
001900     03 FILLER USAGE IS BINARY VALUE ZERO.
002000     03 RETURNED USAGE IS BINARY.
002100 PROCEDURE DIVISION.
002200 MAIN-PARAGRAPH.
002300     ACCEPT START-DATE.
002400*THE NEXT STATEMENT PROPAGATES THE NEGATIVE SIGN ACROSS THE TOP
002500*HALF OF INTEGER-DAYS, AS EXPLAINED IN SECTION 2.2.2.
002600     MOVE HIGH-VALUES TO INTEGER-DAYS.
002700     MOVE ADD-DAYS TO HALFWORD-DAYS.
002800     CALL ''DATE'' USING FUNCTION, START-DATE, INTEGER-DAYS,
002900       END-DATE, RETURN-KODE.
003000     IF RETURNED = 0, DISPLAY ''END-DATE IS '' END-DATE
003100       ELSE DISPLAY ''RETURN-CODE = '' RETURNED.
003200     STOP RUN.
```

DATE Subroutine — A FORTRAN Example

This example gets the current system date and time, and converts a date in Gregorian format to Julian format.

```
      LOGICAL*1 LABEL(45), JDATE(5)
      REAL*8 GDATE
C  THE HD FUNCTION GETS THE DATE AND TIME IN A SPECIFIC FORMAT
      CALL DATE('HD',LABEL)
      WRITE(0,101) LABEL
C  THE GJ FUNCTION CONVERTS DATE IN GREGORIAN TO JULIAN FORMAT
C  THE NEXT STATEMENT SHOWS ANOTHER WAY TO SPECIFY
C  THE VALUE OF THE FIRST ARGUMENT
      ARG1 = 'GJ'
C  THE STARTING DATE IS APRIL 24, 1981 IN GREGORIAN FORMAT
      GDATE = '810424'
      CALL DATE (ARG1, GDATE, JDATE, IRET)
C  TEST RETURN CODE FOR ERRORS
      IF (IRET .EQ. 0) GO TO 1
C  ERROR PROCESSING
      WRITE(0,102) IRET
      GO TO 99
C  NO ERROR IN SUBROUTINE OPERATION
    1 WRITE(0,103) GDATE, JDATE
  101 FORMAT(1X,45A1)
  102 FORMAT(1X, 'ERROR - RETURN CODE = ', I3)
  103 FORMAT(1X, 'GREGORIAN DATE = ', A8/
             1X, 'JULIAN DATE    = ', 5A1/)
   99 PAUSE
      END
```

The output from this program is as follows:

```
   FRIDAY   APRIL 24, 1981           11:31 AM
GREGORIAN DATE = 810424
JULIAN DATE    = 81114

PAUSE:     0
```

DAY

FUNCTION

Computes the day of the week that corresponds to any user-supplied date in the 20th century.

USAGE (arg1, arg2)

Pos	Argument	Type	Size	Comments
arg1	Date	Alpha	6	Provided by the user program, in the format YYMMDD.
arg2	Day of week	Integer	4	Returned by the subroutine. Range from 1 to 7, corresponding to 1=Sunday, 2=Monday, ... 7=Saturday.

DAY Subroutine — A COBOL Example

This program accepts a date in Gregorian format for any day in the 20th century and returns the day of the week as an integer.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DAYC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  GREG-DATE PIC X(6).
000700*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000800*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000900*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001000*BYTES FOR THE INTEGER.
001100 01  DAY-HOLDER.
001200     03 FILLER USAGE IS BINARY VALUE ZERO.
001300     03 DAY-OF-WEEK USAGE IS BINARY.
001400 PROCEDURE DIVISION.
001500 MAIN-PARAGRAPH.
001600     ACCEPT GREG-DATE.
001700     CALL ''DAY'' USING GREG-DATE, DAY-HOLDER.
001800     DISPLAY ''DAY OF WEEK IS '' DAY-OF-WEEK.
001900     STOP RUN.
```

DISMOUNT

FUNCTION

Initiates a dismount of a mounted volume (disk or tape).

USAGE (arg1, ..., arg4)

Pos	Argument	Type	Size	Comments
arg1	Volume	Alpha	6	Name of volume to be dismounted.
arg2	Device Type	Alpha	1	Device type: D = Disk (default) T = Tape Optional. Must be included if arg3 is present.
arg3	Nodisplay Option	Alpha	1	Indicates whether or not to display the dismount screen at the user's workstation: N = No display Blank = Display (default) Optional. If present, arg2 must be included.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-2 below.

NOTE

For FORTRAN programs, the name of this subroutine must be specified as DISMNT.

Table 3-2. DISMOUNT Error Return Codes

Return Code	Meaning
0	Successful dismount.
4	Input volume name blank.
8	Volume not found.
12	Volume cannot be dismounted.
16	Device detached.
20	Volume in use by a user or the operating system.
24	Volume reserved by another user.
28	GETMEM failure (no more segment 0 space).
32	Device reserved by another task.

DISMOUNT Subroutine — A BASIC Example

This program calls the DISMOUNT subroutine to dismount a volume indicated by the user.

```
000100DIM VOLUME$       06
000200DIM DEVICE$       04
000201DEVICE$ = ''DISK''
000202LOOP:
000203GOSUB DISPLAYIT
000204GOSUB DODISMOUNT
000205GOTO LOOP
000210DISPLAYIT:
000360ACCEPT                                                                  !
000410       AT (01,24),                                                      !
000460''Demonstration of DISMOUNT Subroutine'',                               !
000510       AT (07,03),                                                      !
000560''Input the name of the volume that you wish to dismount. The retu!
000610rn code'',                                                              !
000660       AT (08,03),                                                      !
000710''from DISMOUNT will then appear.'',                                    !
000760       AT (10,11),                                                      !
000810''VOLUME ='',                                                           !
000860       AT (10,28), VOLUME$       , CH(06),                            !
000910       AT (11,11),                                                      !
000960''DEVICE ='',                                                           !
001010       AT (11,28), DEVICE$       , CH(04),                            !
001060       AT (11,37),                                                      !
001110''(DISK, TAPE)'',                                                       !
001160       AT (13,11),                                                      !
001210''RETURN CODE ='',                                                      !
001260       AT (13,28), RETURNCODE%   , PIC(##),                           !
001310       AT (16,30),                                                      !
001360''Press ENTER to continue.''
002010RETURN
002100
002110DODISMOUNT
002200   CALL ''DISMOUNT''  ADDR(VOLUME$,DEVICE$,RETURNCODE%)
002210RETURN
```

DISMOUNT Subroutine — A COBOL Example

This sample program calls DISMOUNT to dismount a disk volume called FLOPPY.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-IO. OSMOUNTC.
000300 ENVIRONMENT DIVISION.
000400 OATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  VOLUME-NAME PIC X(6) VALUE ''FLOPPY''.
000700*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000800*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000900*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001000*BYTES FOR THE INTEGER.
001100 01  RETURN-KODE.
001200     03 FILLER USAGE IS BINARY VALUE ZERO.
001300     03 ERROR-CODE USAGE IS BINARY.
001400 PROCEDURE OIVISION.
001500 MAIN-PARAGRAPH.
001600     CALL ''DISMOUNT'' USING VOLUME-NAME, RETURN-KODE.
001700     IF ERROR-CODE NOT EQUAL ZERO DISPLAY ''ERROR-CODE = ''
001800           ERROR-CODE.
001900     STOP RUN.
```

DISMOUNT Subroutine — AN RPG II Example

This program instructs DISMOUNT to dismount the disk volume VOL111. The program checks the subroutine return code and tells the user whether the dismount was successful. The program displays a return code whose value is greater than 0.

```
00100FDISPLAY DD  F                     WS

00200C                    ACCPTSCR1
00201C*
00203C*          *** PREPARE PARAMETERS TO PASS TO RPGCALL MACRO ***
00205C*
00210C                    MOVE 'VOL111'  VOL     6
00220C                    MOVE 'D'       TYPE    1
00240C                    Z-ADD0         RCODE   40
00242C*
00245C*               *** EXIT TO RPGCALL MACRO ***
00247C*
00250C                    EXIT RPGDMT
00255C                    RLABL          VOL
00260C                    RLABL          TYPE
00270C                    RLABL          RCODE
00271C*
00272C*            *** CHECK RETURN CODE ***
00274C*
00275C           RCODE    COMP 0                    99
00280C   99               ACCPTSCR3
00282C  N99               ACCPTSCR2
00284C                    SETON                     LR

00300WSCR1
00400W                 0707       'PRESS ENTER TO DISMOUN'
00500W                 0729       'T DISK VOL111.'
00600WSCR2
00700W                 0707       'DISMOUNT SUCCESSFUL.'
00800W                 0907       'PRESS ENTER TO END JOB'
00900WSCR3
01000W                 0707       'DISMOUNT UNSUCCESSFUL.'
01100W                 0907       'RETURN CODE = '
01200W                 0921RCODE
01300W                 1107       'PRESS ENTER TO END JOB'
```

RPGDMT:

```
        RPGCALL   NAME=RPGDMT,CALL=DISMOUNT,VOL,TYPE,(RCODE,4,F)
```

EXPAND

FUNCTION

Converts a character string from compressed format to external format. EXPAND removes the control characters used to indicate repeated characters and produces text in noncompressed form.

USAGE (arg1, ..., arg5)

Pos	Argument	Type	Size	Comments
arg1	Input	Alpha	var	String to be expanded.
arg2	Input Length	Integer	4	Length of input string. Must be nonnegative and not greater than 2048.
arg3	Output	Alpha	var	Receiver that contains the expanded string.
arg4	Output Length	Integer	4	Maximum length of output string. Must be between 0 and 2048, and is reduced by the subroutine to reflect the actual length of the resulting character string.
arg5	Ret. Code	Integer	4	Error return code. See Table 3-3 below. If the return code is nonzero, the value of the output string is unpredictable.

NOTES

1. The operation of this subroutine is identical to the process used by the XPAND Assembler instruction, used by DMS to expand records.

2. This subroutine is the inverse of the COMPRESS subroutine.

3. The EXPAND subroutine example appears after the description of the COMPRESS subroutine.

Table 3-3. EXPAND Error Return Codes

Return Code	Meaning
0	Successful.
4	Maximum output length too short.
8	Bad compression information was found in the input string.

EXTRACT

FUNCTION

Provides information about the system and the program user. The available information appears below.

USAGE (key1, rec1, key2, rec2, ..., keyn, recn)

The argument list includes keyword-receiver pairs. A keyword must be immediately followed by a receiver. Each keyword selects particular information to be extracted about the system or the user, which the subroutine returns in the receiver. In a few cases, the user program must provide input in part of the receiver.

Each keyword is a 2-byte alpha value. A discussion of keywords, receivers, and the information extracted follows.

Keyword	Recr Type	Recr Size	Receiver Value
A?	Alpha	256	ASCII-to-EBCDIC translation table. Presents EBCDIC characters corresponding to ASCII characters X'00' to X'FF'.
BP	Integer	4	Number of available segment 2 buffer pages.
C	Alpha	16	Cluster information. Bytes 1-2 must contain the device address of the workstation, in binary. The subroutine returns the following information: Byte 1-2 — Device address of archiver diskette (0 if none). 3-16 — Binary zeroes (reserved).
C#	Alpha	4	CPU ID number (CC), and microcode version (MM), in the form CCMM (hexadecimal digits).
CL	Alpha	8	Current program library.
CV	Alpha	6	Current program volume.
D	Alpha	24	Device information. The first byte must contain the device address, in binary. The subroutine fills the receiver with the following: Byte 1 — Device class. 2 — Device type. 3-4 — Usage: EX = Exclusive. SH = Shared. DT = Detached. 5-8 — Task identifier of device owner, or −1 if none. 9-14 — Volume serial number of removable volume (disk or tape only). Blank if nothing mounted. 15-20 — Volume serial number of fixed volume (disk only). Blank if nothing mounted. 21-24 — Binary zeroes (reserved).

Keyword	Recr Type	Recr Size	Receiver Value
D@	Integer	4	Disk I/O count since logon.
DC	Integer	4	Number of devices in the system.
DK	Integer	4	System diskette device number.
DL	Alpha	var	Returns a list of device addresses of the specified device type. The first 2 bytes must contain the device type and the number of device addresses to be returned (specified in binary). Byte 1 — Device type: X'01' = workstation X'02' = magnetic tape X'03' = disk X'04' = printer X'05' = telecommunications Byte 2 — Number of device addresses to be returned (0-253). The receiver size must be at least this value + 2. The receiver contains the following information: Byte 1 — Total number of devices in the specified class. Byte 2 — Number of device addresses supplied. Rest — Device address list (1 byte for each device address).
DV	Alpha	24	Disk volume information. Bytes 1-6 must contain the volume name. The receiver contains the following information: Byte 1 — Device address, or -1 if not mounted. 2 — Volume type: F = Fixed R = Removable Blank = Not mounted 3-4 — Label type: SL = Standard label NL = No label Blank = Not mounted 5-6 — Usage: SH = Shared RR = Restricted removal EX = Exclusive Blank = Not mounted 7-10 — Task identifier of volume mounter, or -1 if none. 11-12 — Blocks per cylinder. 13-14 — Maximum transfer in bytes. 15-16 — Cylinders per volume. 17-18 — Cylinders per physical volume, including bad or unused blocks.

Keyword	Recr Type	Recr Size	Receiver Value
			19-20—Number of files open on this volume. 21-24—Binary zeroes (reserved).
DY	Integer	4	Number of clock units in one day.
E:	Integer	4	Elapsed time in 1/100 seconds.
E?	Alpha	256	EBCDIC-ASCII translation table.
FC	Alpha	1	Default file protection class.
FN	Integer	4	Default printer form number (0-254).
HZ	Integer	4	A/C line frequency.
ID	Alpha	3	Current user's ID.
IL	Alpha	8	Default input library.
IV	Alpha	6	Default input volume.
JC	Alpha	1	Background job default class (A-Z).
JL	Integer	4	Background job default time limit in seconds.
JN	Alpha	8	Background job name.
JS	Alpha	1	Background job default status (R=Run, H=Hold).
L	Alpha	8	Data Link Processor (DLP) status. The first 2 bytes must contain the device address, in binary. The receiver contains the following information: Byte 1—Device status flag: X'80' if open X'40' if reserved Zero otherwise 2-4—Task number of the task that reserved the DLP, zero if device is unreserved. 5-8—Name of the DLP on which the device is SYSGENed.
LI	Integer	4	Default lines-per-page for printer output.
LN	Alpha	38	Data Link Processor (DLP) information. Bytes 1-4 must contain the DLP name. The receiver contains the following information: Byte 1-4—Bit map of devices on DLP. 5-6—First device on DLP. 7—Type of DLP: 1 = 22V06-1 2 = 22V06-2 3 = 22V06-3 8—Number of lines controllable by DLP. 9—Microcode file status: X'00' if stopped X'80' if loaded 10-12—Reserved for future use.

Keyword	Recr Type	Recr Size	Receiver Value
			13-20— Microcode file name, 0 if not loaded. 21-28— Microcode library name, 0 if not loaded. 29-34— Microcode volume name, 0 if not loaded. 35— Reservation status of DLP: X'80' if reserved X'00' if not reserved 36-38— Task number of task that reserved DLP.
ME	Alpha	4	Execute-access mask currently in effect.
MF	Integer	4	Maximum number of files that the user can open, in addition to those already opened.
MR	Alpha	4	Read-access mask currently in effect.
MW	Alpha	4	Write-access mask currently in effect.
NA	Alpha	24	Current user's name (from Userlist).
NR	Integer	4	Total nonresident physical area, in bytes.
O@	Integer	4	Count of "other" I/O transactions (not involving disk, workstation, printer, tape).
OL	Alpha	8	Default output library.
OV	Alpha	6	Default output volume.
P+	Integer	4	Program page-in count.
P−	Integer	4	Program page-out count.
P:	Integer	4	Processor time in 1/100 seconds.
P@	Integer	4	Printer I/O count.
PC	Alpha	1	Default print class (A-Z).
PL	Alpha	8	Default program library (current). See Note 1.
PM	Alpha	1	Default print mode (S, H, K, or O).
PR	Integer	4	Default printer number (for online printing).
PV	Alpha	6	Default program volume (current). See Note 1.
RL	Alpha	8	Run library (initial). See Note 1.
RV	Alpha	6	Run volume (initial). See Note 1.
S#	Alpha	6	System version number, in the form VVRRPP (Version, Revision, Patch).
S+	Integer	4	System page-in count.
S-	Integer	4	System page-out count.
S2	Integer	4	Segment 2 size.
SL	Alpha	8	Default spool library.
SS	Integer	4	Remaining stack space.

Keyword	Recr Type	Recr Size	Receiver Value
SV	Alpha	6	Default spool volume.
T	Alpha	48	Task information. Bytes 1-4 must contain the task number, in binary. The receiver contains the following information: Byte 1 — Workstation device number (binary), -1 if background task. 2-4 — Current user ID for task, blank if none. 5-28 — Current user name for task, blank if none. 29 — Type of task specified: B = Background F = Foreground 30 — Blank. 31-48 — Binary zeroes (reserved).
T#	Integer	4	Task number.
T@	Integer	4	Tape I/O count.
TP	Integer	4	Task priority.
TT	Alpha	1	Task type: F = Foreground B = Background
TV	Alpha	20	Tape volume information. Bytes 1-6 must contain the volume name. The receiver contains the following information: Byte 1 — Device address, -1 if not mounted. 2 — Binary zero (reserved). 3-4 — Density in BPI, in binary: (556, 800, or 1600) 5-6 — Label type: NL = No Label IL = IBM Label AL = ANSI Label Blank = Not mounted 7-8 — Usage: SH = Shared EX = Exclusive Blank = Not mounted 9-12 — Task identifier of tape mounter, -1 if none (in integer(4) format). 13-14 — Current file sequence number (on the tape). 15-20 — Binary zeroes (reserved).
UE	Alpha	4	Default execute-access mask for current user.
UR	Alpha	4	Default read-access mask for current user.
UW	Alpha	4	Default write-access mask for current user.
W#	Integer	4	This workstation's device number, -1 if none.

Keyword	Recr Type	Recr Size	Receiver Value
W@	Integer	4	This workstation's I/O count.
WL	Alpha	8	Default work library.
WV	Alpha	6	Default work volume.
XL	Alpha	8	System library.
XP	Alpha	8	System paging library.
XV	Alpha	6	System volume.
XW	Alpha	8	System work library.

NOTES

1. "Current" refers to the library or volume applicable to the program that contains the EXTRACT call. "Initial" refers to the library or volume applicable to the entire session.

2. For FORTRAN programs, the name of this subroutine must be specified as XTRACT.

EXTRACT Subroutine — A COBOL Example

This program retrieves its own Segment 2 size. This size is always greater than 32767, the maximum size for an integer in COBOL's halfword-binary format. The program circumvents the problem (discussed in Section 2.2.2), by calling the BASIC subroutine 4TO9.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  EXTRACTC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  KEYWORD PIC X(2) VALUE ''S2''.
000700*THE NEXT ITEM RECEIVES THE SEGMENT 2 SIZE FROM EXTRACT AND
000800*PASSES IT TO 4TO9.
000900 01  TEMP PIC X(4).
001000*THE NEXT ITEM RECEIVES THE SEGMENT 2 SIZE FROM 4TO9 AND RETURNS
001100*IT TO THE COBOL PROGRAM.
001200 01  SEG-2-SIZE PIC S9(8).
001300 PROCEDURE DIVISION.
001400 MAIN-PARAGRAPH.
001500     CALL ''EXTRACT'' USING KEYWORD, TEMP.
001600     CALL ''4TO9'' USING TEMP, SEG-2-SIZE.
001700     DISPLAY SEG-2-SIZE.
001800     STOP RUN.
```

EXTRACT Subroutine — A FORTRAN Example

This example calls the EXTRACT subroutine to obtain the user's ID, the default output library, and the number of CPU seconds used. All are displayed on the workstation.

```
C  'OUTLIB' IS THE DEFAULT OUTPUT LIBRARY
C  'ID' IS THE USER'S ID
C  'CPUSEC' IS THE NUMBER OF CPU SECONDS USED
     REAL*8 OUTLIB
     INTEGER*4 ID, CPUSEC
C* CALL EXTRACT (XTRACT IN FORTRAN) WITH ID, OL, AND P: KEYWORDS
     CALL XTRACT ('ID', ID, 'OL', OUTLIB, 'P:', CPUSEC)
C* SINCE CPUSEC RETURNS CPU USAGE IN 1/100 SECS, MUST CONVERT
     SECS = CPUSEC/100.0
     WRITE(0,101) ID, OUTLIB, SECS
 101 FORMAT(1X, 'USER ID IS ', A3/
    1       1X, 'DEFAULT OUTPUT LIBRARY IS ', A8/
    2       1X, 'NUMBER OF CPU SECONDS IS ', F12.2)
     PAUSE
     END
```

EXTRACT Subroutine — AN RPG II Example

This program extracts and displays the user's name and ID, the current device count, the number of files that the user can still open, and the number of system page-ins performed so far.

```
00100FSCREEN  OD  F                    WS

00101C*
00102C*            *** PREPARE PARAMETERS TO BE PASSED ***
00103C*
00110C                    MOVE 'DC'      DC      2
00120C                    Z-ADDO         DCX    40
00200C                    MOVE 'ID'      ID      2
00300C                    MOVE ' '       IDX     3
00400C                    MOVE 'MF'      MF      2
00500C                    Z-ADDO         MFX    40
00600C                    MOVE 'NA'      NA      2
00700C                    MOVE ' '       NAX    24
00800C                    MOVE 'S+'      SP      2
00900C                    Z-A000         SPX    40
00910C*
00920C*           *** EXIT TO THE RPGCALL MACRO ***
00930C*
01000C                    EXIT RPGEXT
01100C                    RLABL          DC
01200C                    RLABL          DCX
01210C                    RLABL          ID
01220C                    RLABL          IDX
01230C                    RLABL          MF
01240C                    RLABL          MFX
01250C                    RLABL          NA
01255C                    RLABL          NAX
01265C                    RLABL          SP
01275C                    RLABL          SPX
01285C*
01295C*          *** DISPLAY EXTRACTED INFORMATION ***
01305C*
01315C                    ENBLEKG
01355C                    ACCPTSCR1
01365C   KG               SETON                  LR

01455WSCR1
01555W              0707       'USER'
01655W              0712NAX
01755W              0738       '(   )'
01855W              0739IDX
01955W              0907       'CURRENT DEVICE COUNT: '
02055W              09300CX
02155W              1107       'YOU MAY OPEN      MORE'
02255W              1129       ' FILES.'
02355W              1120MFX
02455W              1307       'SO FAR,      SYSTEM PA'
```

```
02555W                  1329        'GEINS.'
02655W                  1315SPX
02755W                  2007        'PRESS PF 16 TO EXIT.   '
```

RPGEXT:

```
          RPGCALL    NAME=RPGEXT,CALL=EXTRACT,DC,(DCX,4,F),ID,IDX,MF,     C
                (MFX,4,F),NA,NAX,SP,(SPX,4,F)
```

FIND

FUNCTION

Obtains one or more file, library, or volume names from complete or partial file, library, and volume names supplied by the user program. Also, indicates whether a specified file resides in a specified library and volume.

USAGE (arg1, ..., arg8)

See the note after the argument descriptions for information about specifying the names of files, libraries, and volumes.

Pos	Argument	Type	Size	Comments
arg1	File	Alpha	8	File or files to be found. If blank, a library search is assumed.
arg2	Library	Alpha	8	Library or libraries to be found. If blank, a volume search is assumed.
arg3	Volume	Alpha	6	Volume or volumes to be found. The volume name should not be blank. Only Standard Label (SL) volumes can be searched.
arg4	Starter	Integer	4	Entry at which to begin listing. See Note 3.
arg5	Counter	Integer	4	Maximum number of entries to be listed. The user provides an initial value; the subroutine sets this to the actual count. See Note 3.
arg6	Receiver	Alpha	var	Entries. Each entry is 22 bytes and contains: Byte 1-6—Volume 7-14—Library (can be blank) 15-22—File (can be blank) *Arg8 = A, blank, or omitted*: this is the name or address of the variable that holds the requested entries. *Arg8 = F*: this must be the UFB address (File # in BASIC, or FD in COBOL) of a consecutive file, record size 22, opened in Output or Extend mode.
arg7	File Count	Integer	4	Actual number of eligible entries, returned by the subroutine. Optional, but must be present if arg8 is included. See Note 2.
arg8	Receiver Type	Alpha	1	Type of output to be returned. For alpha receiver (default), specify A or blank. For file receiver, specify F. Optional. If included, arg7 must also be present. See arg6 description.

How to Specify the Names of Files, Libraries, and Volumes

The file, library, and volume arguments can be either standard alphanumeric names, or masks that contain both standard characters and one or more of the special characters ? and *. The significance of these special characters is as follows:

> ? corresponds to *any* string of any length in the name. For example, if Library = ?XYZ?, the subroutine returns all libraries whose names contain the string XYZ preceded and/or followed by any (or no) characters.
>
> * corresponds to a *single* nonblank character in the name. For example, if Library = *, the subroutine returns all one-letter libraries in the specified volume.

Blanks are ignored in the input arguments. Also, a completely blank input argument selects the next level of find. For example, blank file returns a library list, blank file and library returns a volume list.

Examples of File, Library, and Volume Specifications

File	Library	Volume	Items Returned
X	Y	Z	Returns X, Y, and Z if file X exists in library Y on volume Z; otherwise, returns nothing.
*	?	?	All one-letter file names.
?	?ABC?	?	All file names in every library whose name contains ABC.
?	?	VOL123	All files on volume VOL123.
blank	#?PRT	SYSTEM	All print library names on volume SYSTEM.
blank	blank	?	All volume names currently mounted on the system.

NOTES

1. If the subroutine cannot read the VTOC of a volume for any reason, it ignores that volume.

2. Argument 7 provides the total number of entries found, while argument 5 indicates how many entries are to be returned in the receiver. If the program includes argument 7 and if it is larger than argument 5, the subroutine might take more time to execute.

3. The program can use arguments 4 and 5 together to successively output a large number of qualified entries. For example, if Starter=1 and Counter=100, the first 100 entries are returned to the receiver. Then, if Starter is incremented to 101 and Counter remains at 100, a second use of the subroutine results in returning the second 100 entries. Each increment requires a separate call to FIND and adds time to the process.

FIND Subroutine — A COBOL Example

This program allows the user to retrieve the names of files, libraries, or volumes on the system. The program displays output on the workstation.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. FINDC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FILE-NAME  PIC X(8).
000700 77  LIB-RARY  PIC X(8).
000800 77  VOL-UME  PIC X(6).
000900*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001000*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001100*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001200*BYTES FOR THE INTEGER.  TO PASS THE INTEGER TO THE SUBROUTINE,
001300*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001400 01  STARTER.
001500     03 FILLER USAGE IS BINARY VALUE ZERO.
001600     03 START-INTEGER  USAGE IS BINARY VALUE 1.
001700 01  COUNTER.
001800     03 FILLER USAGE IS BINARY VALUE ZERO.
001900     03 COUNT-INTEGER  USAGE IS BINARY.
002000 77  RECEIVER  PIC X(110).
002100 01  ENTRIES.
002200     03 FILLER USAGE IS BINARY VALUE ZERO.
002300     03 ENTRY-COUNT USAGE IS BINARY.
002400 PROCEDURE DIVISION.
002500 FIRST-PARAGRAPH.
002600     ACCEPT FILE-NAME, LIB-RARY, VOL-UME.
002700     IF FILE-NAME = ''!'' GO TO EXIT-PARAGRAPH.
002800*COUNT-INTEGER RECEIVES THE ACTUAL NUMBER OF ENTRIES RETURNED,
002900*IF LESS THAN THE ORIGINAL SPECIFICATION.  THUS IT MUST BE
003000*RE-INITIALIZED FOR THE SUBROUTINE TO BE CALLED AGAIN.
003100     MOVE 5 TO COUNT-INTEGER.
003200 SECOND-PARAGRAPH.
003300     PERFORM CALL-PARAGRAPH.
003400*START-INTEGER IS INCREMENTED EACH TIME THROUGH THE LOOP.  WHEN
003500*IT BECOMES GREATER THAN THE NUMBER OF AVAILABLE ENTRIES, CONTROL
003600*RETURNS TO THE FIRST PARAGRAPH.
003700     IF START-INTEGER GREATER THAN ENTRY-COUNT, MOVE 1 TO
003800       START-INTEGER, PERFORM FIRST-PARAGRAPH.
003900     PERFORM SECOND-PARAGRAPH.
004000 CALL-PARAGRAPH.
004100     MOVE SPACES TO RECEIVER.
004200     CALL ''FIND'' USING FILE-NAME, LIB-RARY, VOL-UME, STARTER,
004300       COUNTER, RECEIVER, ENTRIES.
004400     DISPLAY RECEIVER.
004500     DISPLAY ''ENTRY-COUNT = ''ENTRY-COUNT,
004600           ''  START-INTEGER = ''START-INTEGER,
004700           ''  COUNT-INTEGER = ''COUNT-INTEGER.
004800     ADD 5 TO START-INTEGER.
004900 EXIT-PARAGRAPH.
005000     STOP RUN.
```

FIND Subroutine - A FORTRAN Example

This example finds files, libraries, and volumes on the disk depending on the input that the user enters. The program displays output on the workstation.

```
C  'LIBS' CONTAINS THE NAMES OF LIBRARIES
C  'IFILE', 'ILIB', 'IVOL' ARE ENTERED BY THE USER
C  EVERY RECORD MUST BE 22 BYTES LONG
C  LIBS(22,100) provides 100 RECORDS, EACH 22 BYTES LONG
      LOGICAL*1 LIBS(22,100)
      REAL*8 IFILE, ILIB, IVOL
      ICOUNT = 100
      WRITE(0,103) ' FILE?'
      READ(0,103) IFILE
      WRITE(0,103) ' LIB?'
      READ(0,103) ILIB
      WRITE(0,103) ' VOL?'
      READ(0,104) IVOL
C
C  CALL FIND TO PROVIDE NAMES DEPENDING ON WHAT THE OPERATOR ENTERED
      CALL FIND(IFILE, ILIB, IVOL, 1, ICOUNT, LIBS)
C
      WRITE(0,102) ICOUNT
      DO 10 I=1,5
      WRITE(0,101) (LIBS(J,I),J=1,22)
   10 CONTINUE
  101 FORMAT(1X,22A1)
  102 FORMAT(1X,I5)
  103 FORMAT(A8)
  104 FORMAT(A6)
      PAUSE
      END
```

FLOPIO

FUNCTION

Performs the following I/O operations with a nonlabeled (NL) diskette:

OPEN the diskette as a file
CLOSE the diskette
READ or READ-HOLD from the diskette
WRITE or REWRITE to the diskette
Find the status of a specified diskette

USAGE (arg1, arguments)

Arg1 determines the I/O function that the subroutine performs and the number and nature of the additional arguments.

1. OPEN the diskette as a file

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is OP
arg2	Open Mode	Alpha	2	Mode in which the diskette is open: IN = Input mode IO = IO mode OU = Output mode
arg3	Prname	Alpha	8	User-supplied parameter reference name for the file. Only one file can be open at a time.
arg4	Volume	Alpha	6	Name given the diskette when mounted.
arg5	Record Size	Integer	4	Size of NL diskette records: 4096 for 2200 diskettes (default) 256 for VS/WP and VS diskettes If omitted, the last value used is assumed. See Note 3.
arg6	Ret. Code	Integer	4	Error return code. 0 = Successful open 4 = Not an NL diskette If neither, the subroutine returns the following information from the UFB: Byte 1 — UFBFS2, the second byte of the file status code 2 — UFBXCODE, extended open exit code 3 — UFBF2, open mode flag 4 — Hex '08' Refer to the *VS Operating System Services* for a complete explanation of each of these bytes.

Arguments 3 to 5 are optional. If the program uses an argument, all the previous arguments must be included.

If argument 3 or 4 is omitted or contains only hexadecimal zeroes, the prname and volume names currently in the UFB are moved to these fields.

2. CLOSE the previously opened diskette

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is CL
arg2	Ret. Code	Integer	4	Error return code. See Table 3-4 below. A nonzero value is the file status code for the last WRITE to the file.

3. READ or READ-HOLD from the diskette

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Type of read to be performed: RE = READ RH = READ-HOLD
arg2	Record Number	Integer	4	Sector number to be read. The first sector is 1. A value of 0 is equivalent to a READ NEXT.
arg3	Buffer	Alpha	256	Receiver for the returned record.
arg4	Ret. Code	Integer	4	Error return code. See Table 3-4 below.

4. WRITE or REWRITE to the diskette

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	WRITE or REWRITE: WR = WRITE RW = REWRITE
arg2	Buffer	Alpha	256	Buffer containing the record to be written. See Note 3 for information about its length.
arg3	Ret. Code	Integer	4	Error return code. See Table 3-4 below.

5. Find the status of a specified diskette

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is FI
arg2	Volume	Alpha	6	Name assigned the diskette when mounted. If it contains hexadecimal zeroes or is omitted, the subroutine assumes the volume name currently in the UFB, and replaces the hexadecimal zeroes with that volume name. Must be included if arg3 is present.

Pos	Argument	Type	Size	Comments
arg3	Diskette Status	Alpha	2	I/O status of the diskette relative to the current program, returned by the subroutine: OU = Open for output IN = Open for input IO = Open for I/O CL = Not opened by FLOPIO Optional. If present, arg2 must be included.
arg4	Ret. Code	Integer	4	Error return code: 0 = Diskette found 4 = Not an NL diskette 8 = Diskette not mounted

NOTES

1. Input mode allows READ only. IO mode allows READ, READ-HOLD, and REWRITE. Output mode allows WRITE only.

2. In all cases, an invalid sequence of functions, such as closing an unopened file or doing a READ in input mode, causes the user program to be cancelled.

3. An NL diskette is assumed to have 256-byte sectors. That is the record size used in all READs, REWRITEs, and WRITEs. The size specified for the OPEN command serves only to tell the subroutine whether the sectors are in 2200, VS/WP, or VS order. On VS/WP and VS diskettes, consecutively numbered sectors are physically consecutive and are processed sequentially, starting from the outermost track, 16 sectors per track. On a 2200 diskette, consecutively numbered sectors are located four physical sectors apart within a track. FLOPIO processes the sectors in numeric, rather than physically consecutive, order.

   In Output mode, data is physically written to the diskette in 4096-byte blocks (one track). Therefore, if a multiple of 16 sectors is not written, the unwritten sectors contain undefined data. If this is not desirable, the programmer can use READ/REWRITE in IO mode. This method, however, is noticeably slower.

Table 3-4. FLOPIO Error Return Codes

Return Code	Meaning
0	Successful operation.
10	End-of-diskette encountered (for READ NEXT or READ HOLD).
23	Invalid record number (for READ or READ HOLD).
30	Hardware error.
34	End-of-diskette encountered (for WRITE).

FLOPIO Subroutine - A COBOL Example

This program opens a nonlabeled diskette volume, writes two records to the diskette, closes it, opens it again, reads and displays the two records, and closes the diskette.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. FLOPIOC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION PIC X(2).
000700 77  OPEN-MODE PIC X(2) VALUE ''OU''.
000800 77  PRNAME PIC X(8) VALUE ''FLOPIO''.
000900 77  VOLUME-NAME PIC X(6) VALUE ''FLOPPY''.
001000*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001100*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001200*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001300*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001400*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001500 01  RECORD-SIZE.
001600     03 FILLER USAGE IS BINARY VALUE 0.
001700     03 R-SIZE USAGE IS BINARY VALUE 256.
001800 01  RETURN-KODE.
001900     03 FILLER USAGE IS BINARY VALUE ZERO.
002000     03 ERROR-CODE USAGE IS BINARY.
002100 77  BUF-FER PIC X(256) VALUE SPACE.
002200 01  RECORD-NUMBER.
002300     03 FILLER USAGE IS BINARY VALUE 0.
002400     03 RECORD-COUNTER USAGE IS BINARY.
002500 PROCEDURE DIVISION.
002600 MAIN-PARAGRAPH.
002700     PERFORM OPEN-PARAGRAPH.
002800     PERFORM WRITE-PARAGRAPH VARYING RECORD-COUNTER FROM 1 BY 1
002900       UNTIL RECORD-COUNTER EQUAL 3.
003000     PERFORM CLOSE-PARAGRAPH.
003100     MOVE ''IN'' TO OPEN-MODE.
003200     PERFORM OPEN-PARAGRAPH.
003300     PERFORM READ-PARAGRAPH VARYING RECORD-COUNTER FROM 1 BY 1
003400       UNTIL RECORD-COUNTER EQUAL 3.
003500     PERFORM CLOSE-PARAGRAPH.
003600     STOP RUN.
003700 OPEN-PARAGRAPH.
003800     DISPLAY ''I AM IN THE OPEN-PARAGRAPH.''
003900     MOVE ''OP'' TO FUNCTION.
004000     CALL ''FLOPIO'' USING FUNCTION, OPEN-MODE, PRNAME, VOLUME-NAME,
004100       RECORD-SIZE, RETURN-KODE.
004200     IF ERROR-CODE NOT EQUAL 0 GO TO ERROR-PARAGRAPH.
```

```
004300 WRITE-PARAGRAPH.
004400     DISPLAY ''I AM IN THE WRITE-PARAGRAPH.''
004500     IF RECORD-COUNTER = 1 MOVE ''THE FIRST RECORD'' TO BUF-FER
004600       ELSE MOVE ''THE SECOND RECORD'' TO BUF-FER.
004700     MOVE ''WR'' TO FUNCTION.
004800     CALL ''FLOPIO'' USING FUNCTION, BUF-FER, RETURN-KODE.
004900     IF ERROR-CODE NOT EQUAL ZERO GO TO ERROR-PARAGRAPH.
005000 READ-PARAGRAPH.
005100     DISPLAY ''I AM IN THE READ-PARAGRAPH.''
005200     MOVE ''RE'' TO FUNCTION.
005300     CALL ''FLOPIO'' USING FUNCTION, RECORD-NUMBER, BUF-FER,
005400       RETURN-KODE.
005500     IF ERROR-CODE NOT EQUAL ZERO GO TO ERROR-PARAGRAPH.
005600     DISPLAY BUF-FER.
005700 CLOSE-PARAGRAPH.
005800     DISPLAY ''I AM IN THE CLOSE-PARAGRAPH.''
005900     MOVE ''CL'' TO FUNCTION.
006000     CALL ''FLOPIO'' USING FUNCTION, RETURN-KODE.
006100     IF ERROR-CODE NOT EQUAL ZERO GO TO ERROR-PARAGRAPH.
006200 ERROR-PARAGRAPH.
006300     DISPLAY ''ERROR CODE = '' ERROR-CODE.
006400     STOP RUN.
```

GETPARM

FUNCTION

Provides the ability to generate parameter requests in a higher-level language program.

USAGE

The GETPARM argument list consists of the following sets of arguments. Some are optional, and some are repeatable.

The *GETPARM Definition* argument sequence: GETPARM Type, Form, Prname, PF Key Receiver, Message ID, Message Issuer, Message Line Count, Message Text, Message Text Length

The *Keyword Field type* argument sequence: Specification Type, Keyword, Value, Length, Row Flag, Row, Column Flag, Column, Data Type

The *Text Field type* argument sequence: Specification Type, Text, Text Length, Row Flag, Row, Column Flag, Column

The *PF Key Mask* argument sequence: Specification Type, PF Key Mask

The *ENTER Flag* specification: Specification Type

Each GETPARM argument sequence is described below.

GETPARM Definition Arguments

The following mandatory sequence of nine arguments is included only once in the argument list.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	2	Type of request: I = Specify initial parameters R = Respecify parameter(s) (error correction) ID = Satisfy initial parameters from defaults RD = Satisfy correction parameters from defaults See Note 1 for request type descriptions.
arg2	Form	Alpha	1	Form of screen: A = Acknowledge R = Request S = Select Unless the program specifies a PF key mask (see PFKEY Mask Specification) with the Request and Acknowledge forms, all PF keys are disabled; with the Select form, all PF keys are enabled. See Note 1 for request form descriptions.
arg3	Prname	Alpha	8	Parameter reference name. To satisfy the request via Procedure language statements, prname must be alphanumeric.
arg4	PF Key Receiver	Alpha	1	AID byte. For type ID or RD, indicates key that selects default option. If not used, initialize to @. See Table 3-18 for AID bytes and their meanings.
arg5	Msg ID	Alpha	4	Identifies particular GETPARM screen.
arg6	Msg Iss.	Alpha	6	Identifies source of screen.
arg7	Msg Line Count	Integer	4	Number of lines of message. The message can be specified either as individual lines of text (arg7 nonnegative), or as a single block (arg7 omitted).
arg8	Msg Text	Alpha	var	Message text. *Arg7 specified*: arg8 is an individual line of text, and arg8 and arg9 are repeated for each separate line of text in the message. Each line can begin with one or more of the following control characters: X'5E' (up-arrow) = Center msg text X'5F' (underscore) = Underline msg text X'21' (exclamation pt) = Blink msg text *Arg7 omitted*: arg8 is the entire message text, where lines are separated by an X'0D' character.

Pos	Argument	Type	Size	Comments
arg9	Msg Text Length	Integer	4	Length of message text. A text length of 0, excluding control characters, causes no text line to be generated. If the argument list contains only empty text strings, a single blank is generated as the text.

Keyword Field GETPARM Type

The following argument list defines a single Keyword field, for which the user or Procedure language statements can supply the parameters. The entire set of arguments is specified once for each keyword.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Specifies keyword field type: K/k = Standard keyword field. R/r = Error-respecify keyword field. See Note 2 for uppercase and lowercase usage.
arg2	Keyword	Alpha	8	Keyword name. Can contain any characters, but must be alphanumeric if Procedure language statements specify parameters.
arg3	Value	Alpha	var	Initial value of keyword. Blanks in the field are converted to pseudoblanks on the screen and back to blanks after the user presses a PF key or the ENTER key.
arg4	Length	Integer	4	Length of keyword field. The user specifies zero to process entire field as skip specification (as though arg1 =k or r).
arg5	Row Flag	Alpha	1	Indicates how to position this field: A = Absolute. Rows 9-24 are available, but the row depends on how many lines of message text were displayed. R = Relative (default). Calculated from the "current" row (most recent row displayed), or initial default. Optional.
arg6	Row	Integer	4	Row to display this field. *Arg5=A*: arg6 is actual row. *Arg5=R*: arg6 is number of rows from "current" row. If the user has not specified any fields, current row is (n+8) where n is the number of lines of message text specified (minimum of 1).
arg7	Col. Flag	Alpha	1	Indicates how to position this field: A = Absolute (columns 2-80 are available).

Pos	Argument	Type	Size	Comments
				R = Relative (default). For a new row, "current" column is 2. (Can be 0-78.) C = Center the field in the specified row. J = Right-justify the field in the specified row. Optional.
arg8	Column	Integer	4	Column to display this field. *Arg 7=A:* arg8 is column to display field. *Arg 7=R:* arg8 is number of columns from "current" column. Current column is either 2 (initially, or whenever a row value other than Relative 0 is specified), or the end position of the last field specified plus 1 trailing blank. *Arg 7=C or J:* arg8 is optional, and is ignored if included.
arg9	Data Type	Alpha	1	Data type for this field. Uppercase generates modifiable fields. Lowercase generates protected fields. A/a = Alphanumeric only (A-Z, 0-9, #, @, $). Letters converted to uppercase. C/c = Any character accepted. I/i = Unsigned integers only (0-9). N/n = Numeric only (optional decimal point and sign). L/l = Limited alphanumeric (A-Z, 0-9, #, @, &). Letters converted to uppercase. First character must not be a number. U/u = Any characters. Letters converted to uppercase. H/h = Hexadecimal digits only (0-9, A-F). Numeric and integer fields are limited to 16 characters in length. The VS Procedure language allows the user to override protected fields.

Text Field GETPARM Type

The Text Field type causes text to be displayed on the GETPARM screen. The program specifies the entire argument list once for each line of text to be displayed.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Specifies text field type: T/t = Text field U/u = Underlined text field See Note 2 for upper and lowercase usage.
arg2	Text	Alpha	var	Line of text to be displayed.
arg3	Length	Integer	4	Length of text line (arg2). Specify zero to cause the entire text field spec to be processed as a skip (as though arg1 =t or u).
arg4	Row Flag	Alpha	1	See arg5 of Keyword field type.
arg5	Row	Integer	4	See arg6 of Keyword field type.
arg6	Col. Flag	Alpha	1	See arg7 of Keyword field type.
arg7	Column	Integer	4	See arg8 of Keyword field type.

PF Key Mask Specification

This specification type allows the program to enable/disable any of the 32 PF keys.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Specifies PF Key Mask specification type: P/p = PF Key Mask type See Note 2 for uppercase and lowercase usage.
arg2	PF key mask	Alpha	4	Four byte (32 bit) mask. Each bit corresponds to a PF key: leftmost bit to PF1, rightmost bit to PF32. A bit value of 1 enables the corresponding key, 0 disables the key. For example, the user enables all keys by specifying the value X'FFFFFFFF'. If no mask is supplied, the default action is to disable all PF keys for Acknowledge and Request forms, and to enable all PF keys for the Select form.

ENTER Flag Specification

This specification type allows the user to enable/disable the ENTER key. The default for all form types is to enable the ENTER key.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Indicates ENTER Flag specification type: E/e = Enable ENTER key N/n = Disable ENTER key See Note 2 for upper and lowercase usage.

NOTES

1. The GETPARM request types are 2-byte values, constructed as follows:

   Byte 1 = I, generates an "initial" request, which can be satisfied by a procedure.
   Byte 1 = R, generates a "respecification" (correction) request, which cannot be satisfied by a procedure.

   Byte 2 = blank, the subroutine satisfies the request via the workstation. Byte 2 = D, the subroutine satisfies the request via current ("default") values for the keywords that comprise the request.

   Thus, the combinations work as follows:

   Type = "I ": the subroutine searches for a procedure to satisfy the requested parameters. If they are not found in a procedure, it requests input from the workstation.

   Type = "ID": the subroutine searches for a procedure to satisfy the requested parameters. If none, it uses current values and continues without displaying the request on the workstation.

   Type = "R ": the subroutine satisfies the request from a workstation display.

   Type = "RD": the subroutine satisfies the request from current values only. (This type is not very useful.)

   The following table lists the screen heading that is displayed for each request type and form if workstation input is required.

Type	Form	Heading
I	A	RESPONSE REQUIRED BY PROGRAM name TO ACKNOWLEDGE prname
I	R	INFORMATION REQUIRED BY PROGRAM name TO DEFINE prname
I	S	RESPONSE REQUIRED BY PROGRAM name TO SELECT prname
R	A	CORRECTION REQUIRED BY PROGRAM name TO ACKNOWLEDGE prname
R	R	CORRECTION REQUIRED BY PROGRAM name TO DEFINE prname
R	S	CORRECTION REQUIRED BY PROGRAM name TO SELECT prname

2. Uppercase values cause the field, PF mask, or ENTER flag to be displayed or executed. Lowercase values cause the sequence of arguments of which this argument is a part to be skipped or ignored. Skip specifications allow a program to select particular parameters at runtime without having to generate several similar CALL statements.

3. FORTRAN programs must specify the name of this subroutine as GETPRM.

GETPARM Subroutine — A BASIC Example

This program first displays a screen that requests that the user provide the GETPARM type, form, prname, and the message number, ID, and up to 2 lines of message text. When the user presses the ENTER key, the program displays a GETPARM screen that includes the user-supplied information. It also demonstrates the use of a variety of fields, including alphanumeric, blinking, uppercase, integer, numeric, and protected. The program disables all but PF5.

```
000100DIM TYPE$          02
000200DIM FORM$          01
000300DIM PRNAME$        08
000400DIM PFKEYRECEIVER$ 1
000500DIM MESSAGENO$     04
000600DIM MESSAGEID$     06
000700DIM MESSAGE1$      60
000800DIM MESSAGE2$      60
000900TYPE$ = ''I ''
001000FORM$ = ''S''
001100DIM A$121
001200AGAIN:
001300GOSUB FORMATSCREEN
001400GOSUB DOGETPARM
001500GOTO AGAIN
001600
001700DOGETPARM:
001800A$ = MESSAGE1$ & HEX(0D) & MESSAGE2$
001900CALL ''GETPARM''  ADDR(TYPE$,FORM$,PRNAME$,                              !
002000         PFKEYRECEIVER$,MESSAGENO$,MESSAGEID$,                           !
002100         A$,121%,                                                        !
002200         ''N'',                                                          !
002300         ''P'', HEX(FFFF),                                               !
002400         ''T'',''This is a TEXT FIELD.'',21%,1%,0%,                      !
002500         ''K'',''ALPHANUM'',''THISALPHANUMERICFIELDHASNOBLKS'',30%       !
002600                           ,2%,15%,''A'',                                !
002700         ''R'',''BLINK    '',''This field blinks, allows all characters.'' !
002800                           ,44%,1%,0%,''C'',                             !
002900         ''K'',''UPPER    '',''This is an UPPERCASE FIELD.'',27%,1%      !
003000                          ,0% ,''U'',                                    !
003100         ''K'',''INTEGER  '',''7777788888999996'',16%,1%,0%,''I'',       !
003200         ''K'',''NUMERIC  '',''1234567890.09876'',24%,1%,0%,''N'',       !
003300         ''T'',''SPECIAL PROTECTED OPTION!!!!'',28%,3%,15%,              !
003400         ''K'',''CHARPROT'',''This is a CHARACTER PROTECTED FIELD.''     !
003500                           ,36%,1%,0%,''c'',                             !
003600         ''T'',''ENTER is disabled; only PF5 works.'',34%,2%,0%)
003700RETURN
```

```
003800
003900FORMATSCREEN:
004000ACCEPT                                                                  !
004100       AT (01,24),                                                      !
004200''Demonstration of GETPARM Subroutine'',                                !
004300       AT (06,04),                                                      !
004400''Specify the following parameters, and press ENTER to get a GETPA!
004500RM screen.'',                                                           !
004600       AT (08,03),                                                      !
004700''TYPE:'',                                                              !
004800       AT (08,14), TYPE$          , CH(02),                             !
004900       AT (08,18),                                                      !
005000''(I-initial; R-respecify; ID-initial dflt; RD-respecify dflt)'',       !
005100       AT (09,03),                                                      !
005200''FORM:'',                                                              !
005300       AT (09,14), FORM$          , CH(01),                             !
005400       AT (09,18),                                                      !
005500''(R-request; S-select; A-acknowledge)'',                               !
005600       AT (10,03),                                                      !
005700''PRNAME:'',                                                            !
005800       AT (10,14), PRNAME$        , CH(06),                             !
005900       AT (11,03),                                                      !
006000''MESSAGE #:'',                                                        !
006100       AT (11,14), MESSAGENO$     , CH(04),                             !
006200       AT (12,03),                                                      !
006300''MESSAGEID:'',                                                        !
006400       AT (12,14), MESSAGEID$     , CH(06),                             !
006500       AT (13,03),                                                      !
006600''MESSAGE:'',                                                          !
006700       AT (13,14), MESSAGE1$      , CH(60),                             !
006800       AT (14,14), MESSAGE2$      , CH(60),                             !
006900       AT (18,03),                                                      !
007000''Press ENTER, look at the GETPARM, and see where your parameters       !
007100were placed.''
007200RETURN
```

GETPARM Subroutine — A COBOL Example

This program creates a GETPARM screen that allows the user to specify an output file. The GETPARM is for initial parameters and has the request form. Fields for the file, library, and volume names, with a prompt centered and blinking above them, appear on the screen.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. GETPARMC.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500 FIGURATIVE-CONSTANTS.
000600*THE TWO USER-FIGURATIVE-CONSTANTS ARE CONTROL CHARACTERS FOR THE
000700*GETPARM MESSAGE.
000800     CENTER IS ''5E''
000900     BLINK IS ''21''.
001000 DATA DIVISION.
001100 WORKING-STORAGE SECTION.
001200 77  TY-PE PIC X(2) VALUE ''I''.
001300 77  FO-RM PIC X VALUE ''R''.
001400 77  PR-NAME PIC X(8) VALUE ''OUTPUT''.
001500 77  KEY-RECEIVER PIC X(1).
001600 77  MESSAGE-NUMBER  PIC X(4) VALUE ''9999''.
001700 77  MESS-ENGER PIC X(6) VALUE ''GETPAR''.
001800*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001900*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
002000*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
002100*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
002200*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
002300 01  LINE-COUNT.
002400     03 FILLER USAGE IS BINARY VALUE 0.
002500     03 LINE-OFFSET USAGE IS BINARY VALUE 1.
002600 01  MESS-AGE.
002700     03 CONTROL-1 PIC X VALUE CENTER.
002800     03 CONTROL-2 PIC X VALUE BLINK.
002900     03 TEXT PIC X(27) VALUE ''PLEASE SUPPLY THESE VALUES''.
003000 01  MESSAGE-LENGTH.
003100     03 FILLER USAGE IS BINARY VALUE 0.
003200     03 M-LENGTH USAGE IS BINARY VALUE 29.
003300 77  KEYWORD-TYPE PIC X VALUE ''K''.
003400 77  KEYWORD-1 PIC X(8) VALUE ''FILE''.
003500 77  VALUE-1 PIC X(8) VALUE SPACES.
003600 01  VALUE-LENGTH.
003700     03 FILLER USAGE BINARY VALUE 0.
003800     03 LENGTH USAGE BINARY VALUE 8.
003900 01  ROW-1.
004000     03 FILLER USAGE IS BINARY VALUE 0.
004100     03 ROW-VALUE-1 USAGE IS BINARY VALUE 1.
004200 01  COL-UMN.
004300     03 FILLER USAGE IS BINARY VALUE 0.
004400     03 COLUMN-VALUE USAGE IS BINARY VALUE 10.
```

```
004500 77  DATA-TYPE PIC X(2) VALUE ''L''.
004600 77  KEYWORD-2 PIC X(8) VALUE ''LIBRARY''.
004700 77  VALUE-2 PIC X(8) VALUE SPACES.
004800 01  ROW-2.
004900     03 FILLER USAGE IS BINARY VALUE 0.
005000     03 ROW-VALUE-2 USAGE IS BINARY VALUE 5.
005100 77  KEYWORD-3 PIC X(6) VALUE ''VOLUME''.
005200 77  VALUE-3 PIC X(6) VALUE SPACES.
005300 01  VALUE-3-LENGTH.
005400     03 FILLER USAGE IS BINARY VALUE 0.
005500     03 VOLUME-LENGTH USAGE IS BINARY VALUE 6.
005600 01  ROW-3.
005700     03 FILLER USAGE IS BINARY VALUE 0.
005800     03 ROW-VALUE-3 USAGE IS BINARY VALUE 4.
005900 PROCEDURE DIVISION.
006000 MAIN-PARAGRAPH.
006100     CALL ''GETPARM'' USING TY-PE, FO-RM, PR-NAME, KEY-RECEIVER,
006200         MESSAGE-NUMBER, MESS-ENGER, LINE-COUNT, MESS-AGE,
006300         MESSAGE-LENGTH, KEYWORD-TYPE, KEYWORD-1, VALUE-1,
006400         VALUE-LENGTH, ROW-1, COL-UMN, DATA-TYPE,
006500         KEYWORD-TYPE, KEYWORD-2, VALUE-2, VALUE-LENGTH, ROW-2,
006600         COL-UMN, DATA-TYPE, KEYWORD-TYPE, KEYWORD-3, VALUE-3,
006700         VALUE-3-LENGTH, ROW-3, COL-UMN, DATA-TYPE.
006800     DISPLAY ''VALUE-1 = ''VALUE-1, '' VALUE-2 = ''VALUE-2, '' VALUE-3
006900-        '' = ''VALUE-3.
007000     STOP RUN.
```

GETPARM Subroutine — A FORTRAN Example

This program displays a screen that prompts the user to select the GETPARM type and form, the prname, and message information and text. When the user presses ENTER, the program displays a GETPARM screen with the user-specified information. Text fields demonstrate the use of various fields, including alphanumeric, blinking, forced uppercase, integer, numeric, and protected. The program disables all but PF5.

```
      LOGICAL*1 FORM, LINE1(40), LINE2(40), PFK, PFREC
      INTEGER*2 TYPE
      REAL*8 PRNAME, MISS
      DATA LINE1/40H DEMONSTRATION OF THE GETPARM SUBROUTINE/,
     1     LINE2/40H .......I-TYPE, ACKNOWLEDGE FORM......../,
     2     PFK,PVALUE/'P',ZFFFF0000/,PRNAME/'GPFOR '/,
     3     TYPE,FORM,MID,MISS/'I ','S','0001','GPFOR1'/
C  SET VALUES FOR GETPARM DEFINITION ARGUMENTS
      CALL GTPARM ('I ','S',PRNAME,PFREC,'0001','GPFOR ',2,
     1  ' SPECIFY THE FOLLOWING PARAMETERS',33,
     2  ' THEN PRESS ENTER TO GET A GETPARM SCREEN',41,
     3  'K','TYPE     ',TYPE  ,2,1,0,'A',
     4  'K','FORM     ',FORM  ,1,1,0,'A',
     5  'K','PRNAME   ',PRNAME,6,1,0,'A',
     6  'K','MSG ID   ',MID   ,4,1,0,'C',
     7  'K','MSG ISS  ',MISS  ,6,1,0,'C',
     8  'K','LINE1    ',LINE1,40,1,0,'C',
     9  'K','LINE2    ',LINE2,40,1,0,'C')
      CALL GTPARM (TYPE,FORM,PRNAME,PFREC,MID,MISS,2,
     1  LINE1,40,LINE2,40,
     2  'N',
     3  PFK, PVALUE,
     4  'K','ALPHANUM','LETTERSONLY',11,1,0,'U',
     5  'R','BLINK    ','All characters BLINKING',23,1,0,'C',
     6  'K','UPPER    ','UPPERCASE FIELD',15,1,0,'U',
     7  'K','INTEGER  ','12345678',8,1,0,'I',
     8  'K','NUMERIC  ','12345.78',8,1,0,'N',
     9  'T','SPECIAL PROTECTED OPTION!',25,3,15,
     1  'K','CHARPROT','Char. PROTECTED FIELD',21,1,0,'c',
     2  'T','ENTER DISABLED, ONLY PF5 WORKS',30,2,0)
      PAUSE
      END
```

GETPARM Subroutine — AN RPG II Example

This program creates the GETPARM screen shown below and displays a screen acknowledging the user's input.

```
**************************************************************************

***  MESSAGE 001 BY TEST1

                  RESPONSE REQUIRED BY  PROGRAM  TEST
                          TO SELECT OPTIONS

ENTER FILE INFORMATION AND PRESS PF5 TO DEFINE INPUT, OR

PRESS PF16 TO END JOB.


     FILE     = ********
     LIBRARY  = ********
     VOLUME   = ******

**************************************************************************
```

```
00100FDISPLAY DD  F                    WS

00100C*           *** PREPARE PARAMETERS TO PASS TO RPGCALL MACRO ***
00150C*
00200C                    MOVE 'I '      TYPE    2
00300C                    MOVE 'S'       FORM
00400C                    MOVE 'OPTIONS 'PRNAME  8
00500C                    MOVE ' '       PFK     1
00600C                    MOVE '001 '    MSGID   4
00700C                    MOVE 'TEST1 '  MSGIS   6
00702C*
00704C*  *** USE TEMPORARY VARIABLES TO BUILD MESSAGE LONGER THAN 8 BYTES ***
00706C*
00710C                    MOVEL'ENTER FI'TEMP1  16
00720C                    MOVE 'LE INFOR'TEMP1
00730C                    MOVEL'MATION A'TEMP2  16
00740C                    MOVE 'ND PRESS'TEMP2
00750C                    MOVELTEMP1     HOLD1  32
00760C                    MOVE TEMP2      HOLD1
00770C                    MOVEL' PF5 TO 'TEMP1
00780C                    MOVE 'DEFINE I'TEMP1
00790C                    MOVEL'NPUT, OR'TEMP3   8
00791C                    MOVELTEMP1     HOLD2  24
```

```
00792C                    MOVE TEMP3      HOLD2
00793C                    MOVELHOLD1      MSG     56
00794C                    MOVE HOLD2      MSG
00900C                    Z-ADD56         MSGLN   40
01000C                    MOVE 'T'        T        1
02000C                    MOVEL'PRESS PF' TEMP1
02100C                    MOVE '16 TO EN' TEMP1
02200C                    MOVE 'D JOB.  ' TEMP3
02300C                    MOVELTEMP1      TEXT    24
02400C                    MOVE TEMP3      TEXT
02900C                    Z-ADD24         TEXTLN  40
03000C                    Z-ADD0          ROWSK   40
03100C                    Z-ADD0          COLSK   40
03200C                    MOVE 'K'        K1       1
03300C                    MOVE 'FILE    ' KEY1     8
03400C                    MOVE ' '        VAL1     8
03500C                    Z-ADD8          LEN1    40
03600C                    Z-ADD3          ROWSK1  40
03700C                    Z-ADD5          COLSK1  40
03800C                    MOVE 'A'        TYPE1    1
03810C                    MOVE 'K'        K2       1
03900C                    MOVE 'LIBRARY ' KEY2     8
04000C                    MOVE ' '        VAL2     8
04100C                    Z-ADD8          LEN2    40
04200C                    Z-ADD1          ROWSK2  40
04300C                    Z-ADD5          COLSK2  40
04400C                    MOVE 'A'        TYPE2    1
04410C                    MOVE 'K'        K3       1
04500C                    MOVE 'VOLUME  ' KEY3     8
04600C                    MOVE ' '        VAL3     6
04700C                    Z-ADD6          LEN3    40
04800C                    Z-ADD1          ROWSK3  40
04900C                    Z-ADD5          COLSK3  40
05000C                    MOVE 'A'        TYPE3    1
05100C                    MOVE 'P'        P        1
05102C*
05104C*     *** PREPARE PF KEY MASK USING BITON AND BITOF ***
05105C*          *** (ENABLE PF 5 AND 16 ONLY) ***
05106C*
05110C                    BITON'4'        PM1      1
05120C                    BITOF'0123567'  PM1
05121C                    BITON'7'        PM2      1
05122C                    BITOF'0123456'  PM2
05123C                    BITOF'01234567' PM3      1
05124C                    BITOF'01234567' PM4      1
05130C                    MOVELPM1        PM5      2
05140C                    MOVE PM2        PM5
05150C                    MOVELPM3        PM6      2
05155C                    MOVE PM4        PM6
05160C                    MOVELPM5        PMASK    4
05165C                    MOVE PM6        PMASK
```

```
05170C*
05175C*     *** PMASK IS THE PF KEY MASK; EFLAG IS THE ENTER FLAG ***
05180C*
05300C                    MOVE 'N'       EFLAG   1
05310C*
05320C*                 *** EXIT TO RPGCALL MACRO ***
0530C*
05400C                    EXIT RPGGET
05500C                    RLABL          TYPE
05600C                    RLABL          FORM
05700C                    RLABL          PRNAME
05800C                    RLABL          PFK
05900C                    RLABL          MSGID
06000C                    RLABL          MSGIS
06100C                    RLABL          MSG
06200C                    RLABL          MSGLN
06300C                    RLABL          T
06400C                    RLABL          TEXT
06500C                    RLABL          TEXTLN
06600C                    RLABL          ROWSK
06700C                    RLABL          COLSK
06800C                    RLABL          K1
06810C                    RLABL          K2
06820C                    RLABL          K3
06900C                    RLABL          KEY1
07000C                    RLABL          KEY2
07100C                    RLABL          KEY3
07200C                    RLABL          VAL1
07300C                    RLABL          VAL2
07400C                    RLABL          VAL3
07500C                    RLABL          LEN1
07600C                    RLABL          LEN2
07700C                    RLABL          LEN3
07800C                    RLABL          ROWSK1
07900C                    RLABL          ROWSK2
08000C                    RLABL          ROWSK3
08100C                    RLABL          COLSK1
08200C                    RLABL          COLSK2
08300C                    RLABL          COLSK3
08400C                    RLABL          TYPE1
08500C                    RLABL          TYPE2
08600C                    RLABL          TYPE3
08700C                    RLABL          P
08710C                    RLABL          PMASK
08720C                    RLABL          EFLAG
08722C*
08724C*     *** IF PF 16 WAS PRESSED, END JOB; OTHERWISE, ACKNOWLEDGE ***
08725C*                       *** USER'S INPUT ***
08727C*
08730C           PFK      COMP 'P'                          99
08820C  N99               ACCPTSCR1
08830C                    SETON                        LR
```

```
08920WSCR1
09020W                  0707        'INPUT FILE IS'
09120W                  0721VAL1
09220W                  0807        'IN LIBRARY'
09320W                  0821VAL2
09420W                  0907                        'ON VOLUME'
09520W                  0921VAL3
09620W                  1205                        'PRESS ENTER TO END JOB'
```

```
RPGGET:

        RPGCALL   NAME=RPGGET,CALL=GETPARM,TYPE,FORM,PRNAME,PFK,       C
              MSGID,MSGIS,MSG,(MSGLN,4,F),T,TEXT,(TEXTLN,4,F),         C
              (ROWSK,4,F),(COLSK,4,F),K1,KEY1,VAL1,(LEN1,4,F),         C
              (ROWSK1,4,F),(COLSK1,4,F),TYPE1,K2,KEY2,VAL2,(LEN2,4,F),C
              (ROWSK2,4,F),(COLSK2,4,F),TYPE2,K3,KEY3,VAL3,(LEN3,4,F),C
              (ROWSK3,4,F),(COLSK3,4,F),TYPE3,P,PMASK,EFLAG
```

HEXPACK

FUNCTION

Converts a string of hexadecimal digits to its ASCII character equivalent.

USAGE (arg1, ..., arg3)

Pos	Argument	Type	Size	Comments
arg1	Hex digits	Alpha	var	String of hexadecimal digits to be converted.
arg2	Receiver	Alpha	var	String to receive the ASCII characters. The length of this string must be at least half the length of the input string.
arg3	Length	Integer	4	Length of input string. If odd, the program ignores the last character of the input string.

NOTES

1. This subroutine is equivalent to the BASIC language HEXPACK statement.
2. The subroutine does not check for valid hexadecimal digits.
3. For FORTRAN programs, the name of this subroutine must be specified as HXPACK.

HEXPACK Subroutine — A FORTRAN Example

This example converts a user-supplied hexadecimal character string into its ASCII equivalent. Both are displayed on the screen.

```
      LENGTH = 4
      WRITE(0,101) ' ENTER 4 HEX CHARS'
C  'HCHARS' CONTAINS 4 HEXADECIMAL CHARACTERS TO BE CONVERTED
      READ(0,102) HCHARS
C  STOP IF USER ENTERS 9999
      IF(HCHARS.EQ.'9999') GO TO 99
C
C  'ACHARS' CONTAINS THE ASCII STRING THAT CORRESPONDS TO HCHARS
C  CALL HEXPACK (HXPACK IN FORTRAN) TO PERFORM THE CONVERSION
      CALL HXPACK (HCHARS, ACHARS, LENGTH)
C
      WRITE(0,103) HCHARS, ACHARS
  101 FORMAT(A20)
  102 FORMAT(A4)
  103 FORMAT(1X, Z8, 5X, A2)
   99 PAUSE
      END
```

HEXUNPK

FUNCTION

Converts a string of ASCII characters into hexadecimal digits.

USAGE (arg1, ..., arg3)

Pos	Argument	Type	Size	Comments
arg1	ASCII st.	Alpha	var	String of ASCII characters to be converted.
arg2	Receiver	Alpha	var	String to receive the hexadecimal characters. The length of this string must be at least twice the length of arg1.
arg3	Length	Integer	4	Length of the input string.

NOTES

1. This subroutine is equivalent to the BASIC language HEXUNPACK statement.

2. For FORTRAN programs, the name of this subroutine must be specified as HXUNPK.

HEXUNPK Subroutine — A FORTRAN Example

This example converts an ASCII string entered by the user into its hexadecimal equivalent. Both are displayed on the screen.

```
C  'ALPHA' CONTAINS UP TO 5 ASCII CHARACTERS
C  'HEX' IS ITS HEXADECIMAL EQUIVALENT
      LOGICAL*1 ALPHA(5), HEX(10)
      WRITE(0,101) ' ENTER LENGTH, STRING'
      READ(0,102) LENGTH, (ALPHA(I), I=1,LENGTH)
C  USER ENTERS * TO STOP
      IF(ALPHA(1) .EQ. 1H*) GO TO 99
C
C  CALL HEXUNPK (HXUNPK IN FORTRAN) TO PERFORM CONVERSION
      CALL HXUNPK(ALPHA, HEX, LENGTH)
C
      WRITE(0,103) HEX
  101 FORMAT(A21)
  102 FORMAT(I1, 5A1)
  103 FORMAT(1X, 10A1)
   99 PAUSE
      END
```

LINK

FUNCTION

Allows the user to link to a program or procedure and to specify a cancel exit for the link. The user program can also specify any arguments that are needed to execute the linked program or procedure.

USAGE (arg1, ..., arg15)

Pos	Argument	Type	Size	Comments
arg1	Program	Alpha	8	Program or procedure to be linked to.
arg2	Link Type	Alpha	1	Where program to be linked resides: S = Check system only P = Use library/volume named in arg3 and arg4 Blank = Use program library and volume associated with the user
arg3	Library	Alpha	8	Library (must be included, but is ignored unless arg2=P).
arg4	Volume	Alpha	6	Volume (must be included, but is ignored unless arg2=P).
arg5	Argument Count	Integer	4	Number of arguments to be passed to the program. See arg6. This value can be 0.
arg6	Arg(s)	Variable		Argument(s) to be passed to the linked program. Arg5 specifies the number of times this argument is repeated. If arg5=0, this argument must be omitted. The length and type of this argument depend on the requirements of the linked program.
arg7	Cancel Exit Flag	Alpha	1	Cancel exit option: C = Cancel exit only N = Cancel exit, allow no debug processing D = Cancel exit, allow no debug processing but generate full dump Blank = No special exit processing
arg8	Message	Alpha	var	Message to override PF16 text. Ignored if arg7 is blank. Maximum length is 27 characters.
arg9	Message Length	Integer	4	Length of PF16 message (arg8). Specify zero for no PF16 message override.
arg10	HELP Disable Flag	Alpha	1	HELP key disable/enable: N = Disable HELP H or blank = Enable HELP

Pos	Argument	Type	Size	Comments
arg11	PF Key Mask	Alpha	2	32-bit mask to enable/disable Command Processor PF keys. This feature is not currently implemented in the operating system.
arg12	Cancel Receiver	Alpha	var	Receiver for the cancel exit information list. Ignored if arg7 is blank.
arg13	Cancel Receiver Length	Integer	4	Maximum length of arg12. Must be nonzero. Register and other information require 128 bytes; the remainder of arg12 contains as much of the cancel message list as fits into the value of arg13 minus 128.
arg14	Completion Code	Integer	4	Indicates the result of the link: 0 = Successful link 8 = Unsuccessful link 16 = Program canceled
arg15	Ret. Code	Integer	4	Error return code. *Arg14=0:* This is the return code from the linked program. *Arg14=8:* See Table 3-5 below. *Arg14=16:* This field is not set.

NOTE

Arguments 2 and 5 through 13 are optional; however, if any of them is included, all preceding arguments must be present. Several arguments must be present in pairs, whether or not both are used (args 8 and 9, and args 12 and 13). If arg5 is zero, arg6 must be omitted, and arguments 14 and 15 are both required.

Table 3-5. LINK Error Return Codes

Return Code	Meaning
0	Not a program file, and the procedure interpreter cannot be invoked.
4	Volume not mounted.
8	Volume in exclusive use by another user.
12	All buffers in use when one was required.
16	Directory not found.
20	File not found.
24	(Unused).
28	Access to program's file-protection class denied.
32	FDX1 and FDX2 conflict detected by READFDR.
36	FDX2 and FDR conflict detected by READFDR.
40	Invalid parameter passed to READFDR (including NL volume type).
44	I/O error on VTOC.
48	Unable to read FDR2 record (additional extent specifications).
52	Invalid program file; unable to complete link.
56	File open other than shared read-only.

LINK Subroutine — A COBOL Example

This program links to the EDITOR dynamically. It also specifies an exit option that returns to the program rather than to the Command Processor if the linked-to program is cancelled.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. LINKC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  LINKNAME PIC X(8) VALUE ''EDITOR''.
000700 77  LOCATION PIC X(1) VALUE ''S''.
000800*SINCE THE LINK TYPE IS ''S'', THE NEXT TWO ARGUMENTS ARE IGNORED
000900*THOUGH THEY MUST BE CODED.
001000 77  LIB-RARY PIC X(8).
001100 77  VOL-UME  PIC X(6).
001200*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001300*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001400*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001500*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001600*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001700 01  PARAMETERS.
001800     03 FILLER USAGE IS BINARY VALUE 0.
001900     03 PARAMETER-COUNT USAGE IS BINARY VALUE 0.
002000 77  EXIT-OPTION  PIC X VALUE ''C''.
002100 77  PF16-MESSAGE  PIC X(16) VALUE ''RETURN TO LINKC!''.
002200 01  MESSAGE-LENGTH.
002300     03 FILLER BINARY VALUE 0.
002400     03 FILLER BINARY VALUE 16.
002500 01  COMPLETION.
002600     03 FILLER USAGE BINARY VALUE ZERO.
002700     03 COMPLETION-CODE USAGE BINARY.
002800 01  ERRORS.
002900     03 FILLER USAGE BINARY VALUE ZERO.
003000     03 ERROR-CODE USAGE BINARY VALUE ZERO.
003100 PROCEDURE DIVISION.
003200 MAIN-PARAGRAPH.
003300     CALL ''LINK'' USING LINKNAME, LOCATION, LIB-RARY, VOL-UME,
003400          PARAMETERS, EXIT-OPTION,
003500          PF16-MESSAGE, MESSAGE-LENGTH,
003600          COMPLETION, ERRORS.
003700     DISPLAY ''THE COMPLETION CODE IS ''COMPLETION-CODE,
003800       '' THE RETURN CODE IS ''ERROR-CODE.
003900     STOP RUN.
```

LINK Subroutine — AN RPG II Example

This program allows the user to update the records of File A or to run the SORT utility. When the user presses PF1 from Screen 1 (SCR1), the program calls LINK and links to the SORT utility. If the user interrupts SORT with the HELP key, the cancel exit message supplied here ("RESUME UPDATING FILE A") replaces the usual Command Processor PF16 message. The program checks the return code and the completion code and displays them if they are nonzero.

```
00100FFILEA   UC  F      09R04AI     1 DISK
00200FDISPLAY DD  F                    WS

00400IFILEA   AA  01
00500I                                        1   40KEYA
00600I                                        5    9 INFOA

00610C*
00620C*                  *** DISPLAY MENU ***
00630C*
00700C           MENU      TAG
00800C                     ENBLEK0,KG,K1
00900C                     ACCPTSCR1
00910C                     SETOF                     99
00920C*
00930C*          *** END JOB OR GO TO WHERE LINK IS PERFORMED OR ***
00940C*                 ***  READ IN A RECORD TO UPDATE ***
00950C*
01000C   KG               SETON                     LR
01010C   KG               GOTO END
01020C   K1               GOTO SORT
01100C           KEYA      CHAINFILEA
01110C*
01120C*            *** DISPLAY AND UPDATE RECORD ***
01130C*
01200C                     ENBLEK0,K1
01300C                     ACCPTSCR2
01400C   K0                EXCPT
01500C                     GOTO MENU
01510C*
01520C*        *** PREPARE PARAMETERS FOR LINK TO SORT UTILITY ***
01530C*
01600C           SORT      TAG
01720C                     MOVE 'SORT    'PROG   8
01800C                     MOVE 'S'       TYPE   1
01900C                     MOVE 'DUMMY   'LIBR   8
01910C                     MOVE 'DUMMY   'VOLM   6
01920C                     Z-ADD0         PCNT  40
01930C                     MOVE 'C'       CEXT   1
02000C                     MOVEL'RESUME U'DUM   16
02005C                     MOVE 'PDATING 'DUM
02010C                     MOVELDUM       MSG   22
02015C                     MOVE 'FILE A'  MSG
```

```
02040C                    Z-ADD22         MSGLN   40
02050C                    Z-ADD0          CCODE   40
02055C                    Z-ADD0          RCODE   40
02060C*
02065C*            *** EXIT TO RPGCALL MACRO ***
02070C*
02100C                    EXIT RPGLNK
02200C                    RLABL           PROG
02210C                    RLABL           TYPE
02300C                    RLABL           LIBR
02400C                    RLABL           VOLM
02500C                    RLABL           PCNT
02600C                    RLABL           CEXT
02700C                    RLABL           MSG
02710C                    RLABL           MSGLN
02711C                    RLABL           CCODE
02800C                    RLABL           RCODE
02801C*
02802C*          *** CHECK RETURN CODES ***
02803C*
02810C          CCODE     COMP 0                    99
02900C          RCODE     COMP 0                    99
03000C                    GOTO MENU
03110C          END       TAG

032000FILEA   E
033000                        KEYA       4
034000                        INFOA      9

03700WSCR1
03720W       99       B0107          'ERROR IN SORT REQUEST'
03730W       99       B0215          'RETURN CODE = '
03740W       99       B0315          'COMPLETION CODE = '
03750W       99       B0930RCODE
03760W       99       B1035CCODE
03800W                 0507           'ENTER THE NUMBER OF TH'
03900W                 0529           'E RECORD YOU WISH TO U'
04000W                 0551           'PDATE,'
04100W                 1015                                  KEYA   40
04110W                 1207           'OR PRESS PF 1 TO RUN T'
04120W                 1229           'HE SORT UTILITY,'
04200W                 1607           'OR PRESS PF 16 TO END '
04300W                 1629           'THE JOB.'
04500WSCR2
04600W                 0507           'MAKE CHANGES AND PRESS'
04700W                 0530           'ENTER TO UPDATE THIS R'
04800W                 0552           'ECORD,'
04900W                 0707           'OR PRESS PF 1 TO EXIT.'
05000W                1215KEYA
05100W                1315INFOA                             INFOA
```

RPGLNK:

```
      RPGCALL    NAME=RPGLNK,CALL=LINK,PROG,TYPE,LIBR,VOLM,          C
            (PCNT,4,F),CEXT,MSG,(MSGLN,4,F),(CCODE,4,F),(RCODE,4,F)
```

LOADCODE

FUNCTION

Allows the user to load specified microcode into a device.

USAGE (arg1, ..., arg14)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	The load function to be performed: C = Load configuration table D = Load device P = Load peripheral processor
arg2	TC Line Name	Alpha	8	New TC line name. Specify X'00' if none (default). Must be present if arg1=P.
arg3	Device Nr.	Integer	4	Number of the device to be loaded.
arg4	Load Type	Alpha	1	Indicates the type of load to be done: T = Load by type N = Load by name U = Unload to default I = Interrupt-driven ("load current")
arg5	Microcode Type	Integer	4	Microcode type ID number. Ignored if arg4 = U or I.
arg6	File name	Alpha	8	File name for load-by-name. Must be present if arg4 = N, otherwise ignored.
arg7	Library Name	Alpha	8	Library name for file named in arg6. Must be present if arg4 = N, otherwise ignored. If X'00', the default microcode library is used.
arg8	Volume Name	Alpha	6	Volume name for file named in arg6. Must be present if arg4 = N, otherwise ignored. If X'00', the default system volume is used.
arg9	Start Location	Integer	4	Starting location in the specified device to be loaded. Default = 0. Ignored if arg4 = I.
arg10	Code Length	Integer	4	Length of microcode to be loaded. If 0 or omitted, the entire microcode file is loaded. Ignored if arg4 = I.
arg11	Condition Flag	Alpha	1	Indicates whether to perform the load if the desired microcode is already loaded: C = Load conditionally (default) U = Load unconditionally. Ignored if arg4 = I.
arg12	Renew Option	Alpha	1	Indicates whether code is to be renewed on DLP/PP error (interrupt-driven call): R = Renewable microcode (default) N = Nonrenewable microcode Optional.

Pos	Argument	Type	Size	Comments
arg13	Interrupt Flag	Alpha	1	Indicates whether task or system is to handle power-on/HELP interrupts: S = System handling (default) T = Task handling Optional.
arg14	Ret. code	Integer	4	Error return code. See Table 3-6 below.

NOTE

For FORTRAN programs, the name of this subroutine must be specified as LOADCD.

Table 3-6. LOADCODE Error Return Codes

Return Code	Meaning
0	Successful load.
4	Device/PP specified cannot be programmed.
8	Specified microcode file not found. (Also set when specified class and type of microcode are not included in UCB MC list, or when specified file name is not a valid alphanumeric string.)
12	Device/PP not reserved exclusively by the caller.
16	Error in opening microcode file, or file not consecutive.
20	I/O error when reading microcode file.
24	One of the following errors: 1. I/O error while loading device or PP microcode, or configuration tables; 2. Error when restarting device or PP after loading microcode; 3. Unable to load device because PP code is missing, or attempt to load PP fails for any reason; 4. Unable to load PP code because configuration tables are missing, or attempt to load tables fails for any reason.
28	Insufficient memory pool (GETMEM failure).
32	(Reserved)
36	Incompatible options: 1. UNLOAD and LOAD-BY-NAME both specified. 2. CLOAD and INTERRUPT both specified.
40	Other devices on cluster not all reserved by the calling task (non-interrupt-driven LOADCODE only).

LOADCODE Subroutine — A COBOL Example

This program loads microcode for a serial workstation to a combined workstation.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. LOADCDEC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION PIC X(2) VALUE ''D''.
000700*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000800*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000900*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001000*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001100*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001200 01  DEVICE.
001300     03 FILLER USAGE IS BINARY VALUE ZERO.
001400     03 DEVICE-NUMBER USAGE IS BINARY VALUE 3.
001500 77  LOAD-TYPE PIC X(1) VALUE ''N''.
001600 01  CODE-TYPE.
001700     03 FILLER USAGE IS BINARY VALUE 0.
001800     03 CODE-ID  USAGE IS BINARY VALUE 11.
001900 77  FILE-NAME PIC X(8) VALUE ''.MC2246S''.
002000 77  LIB-RARY PIC X(8) VALUE ''.SYSTEM.''.
002100 77  VOL-UME PIC X(6) VALUE ''OS''.
002200 77  START-ADDRESS PIC X(6) VALUE 0.
002300 01  RETURNCODE.
002400     03 FILLER USAGE IS BINARY VALUE 0.
002500     03 ERROR-CODE USAGE IS BINARY.
002600 PROCEDURE DIVISION.
002700 MAIN-PARAGRAPH.
002800     CALL ''LOADCODE'' USING FUNCTION, DEVICE, LOAD-TYPE, CODE-TYPE,
002900          FILE-NAME, LIB-RARY, VOL-UME, START-ADDRESS, RETURNCODE.
003000     IF ERROR-CODE NOT EQUAL 0 DISPLAY ''ERROR-CODE = ''ERROR-CODE,
003100       ELSE DISPLAY ''THE NEXT SCREEN WILL BE BLANK. ONLY THE HEL
003200-      ''P KEY WILL BE ENABLED.''.
003300*WHEN THE ENTER KEY IS PRESSED A BLANK SCREEN WILL APPEAR AND ALL
003400*KEYS BUT THE HELP KEY WILL BE DISABLED.  PRESS HELP AND THEN
003500*PRESS PF KEY 1 (CONTINUE) FROM THE COMMAND PROCESSOR MENU.  THE
003600*NEXT SCREEN WILL BE BLANK WITH ONLY THE CURSOR POSITION, ENTER
003700*AND HELP KEYS ENABLED.  PRESS ENTER TO CONTINUE.
003800     DISPLAY ''PRESS ENTER TO TERMINATE THE PROGRAM.''.
003900     STOP RUN.
```

LOGOFF

FUNCTION

Terminates the user program and logs the user off.

USAGE No arguments are required.

NOTE

If the user program containing the reference to this subroutine is run from a program with a Cancel Exit option, the subroutine terminates the program but does not log the user off.

LOGOFF Subroutine — A BASIC Example

This example simply calls the LOGOFF subroutine to terminate processing and log the user off.

```
000100CALL ''LOGOFF''
```

MESSAGE

FUNCTION

Allows communication of messages between workstations (tasks).

Each user who is to receive messages must create a "port" (analogous to a mailbox). The user assigns the port a name, which is used to send messages to the creator of the port. The port is also assigned a buffer size, which is the maximum total size of all messages not read ("checked") by the port's creator.

Users can then transmit messages to that port and the port's creator can check for them. The various options for the transmit and check processes are discussed in the appropriate sections below.

USAGE (arg1, arguments)

Arg1 defines the message function and determines the number and nature of the remaining arguments.

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Type of message function: CR = Create message port DE = Destroy message port XM = Transmit message XW = Transmit message and wait if buffer is full CH = Check message port for message

The remaining arguments depend on the function.

1. Create a message port

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is CR
arg2	Port Name	Alpha	4	Name of port to be created.
arg3	Buffer Size	Integer	4	Maximum cumulative message size assigned to this port (1-2014). Optional. Default is 2014. When the user checks the messages, the cumulative message size is reduced.
arg4	Ret. Code	Integer	4	Error return code: 0 = Successful creation of port 4 = Another task is using this port 8 = This task is using this port

2. Destroy a message port

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is DE
arg2	Port Name	Alpha	4	Name of port to be destroyed.
arg3	Ret. Code	Integer	4	Error return code: 0 = Successful. 4 = Successful, but 1 or more waiting messages were not received and have been lost. 8 = No such port was created by this task.

If there are any messages in the port, they are lost when the user destroys the port. It might be appropriate to check the port for messages before destroying it.

3. Transmit a message

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is either XM or XW. With the XW function (Transmit-and-Wait), the screen is locked until the receiving port is checked and this message is received.
arg2	Port Name	Alpha	4	Name of port to which the program transmits the message.
arg3	Message	Alpha	var	Message to be sent.
arg4	Msg Length	Integer	4	Length of the message in characters.
arg5	Ret. Code	Integer	4	Error return code: 0 = Message queued. 4 = Port named has not been created. 8 = For arg1=XM only. Unable to insert message into the message buffer of the receiving port because the buffer is full. 12 = Port named can only be used by privileged code.

4. Check message port for message

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is CH
arg2	Port name	Alpha	4	Name of port to be checked.
arg3	Check Type	Alpha	1	Type of check to perform: W = Check and wait until message is received T = Check and wait until message is received or time interval (arg4) has expired. K = Check and wait until message is received or a PF or ENTER key is pressed B = Check and wait until message is received, key is pressed, or time expires. For K and B options, if the workstation keyboard is locked, a return code of 12 results.
arg4	Time Interval	Integer	4	Time to wait in hundredths of a second. Applicable for check types T and B.
arg5	Message Receiver	Alpha	var	Receiver for message. Its length must be at least the value of arg6.
arg6	Message Length	Integer	4	Length of message receiver. This is the maximum length to be returned. The subroutine reduces this value to reflect the actual message length. If the message is longer, it is truncated.
arg7	Ret. Code	Integer	4	Error return code: 0 = Message received 8 = Time interval expired 12 = Keyboard locked, probably by PF or ENTER key being pressed 16 = No such port was created by this task, or check was canceled (for arg3 = T)

NOTE

For FORTRAN programs, the name of this subroutine must be specified as MESAGE.

MESSAGE Subroutine — A BASIC Example

This example sets up a message port and demonstrates how the subroutine passes messages between workstations.

```
000100DIM TYPE$          02
000200DIM PORTNAME$      04
000300DIM MESSAGE1$      66
000400DIM MESSAGE2$      66
000410DIM MESSAGE3$     132
000500DIM CHECKTYPE$     01
000501CHECKTYPE$ = ''T''
000510LOOP:
000520GOSUB PUTSCREEN
000530GOSUB DOMESSAGE
000540GOTO LOOP
000541
000550PUTSCREEN:
000600ACCEPT                                                                  !
000700       AT (01,12),                                                      !
000800''Demonstration of Sending Messages through MESSAGE Subroutine'',       !
000900       AT (03,03),                                                      !
001000''Fill in the following information to either (1) create a message      !
001100 port,'',                                                               !
001200       AT (04,03),                                                      !
001300''(2) destroy a message port, (3) transmit to a port (either retur      !
001400n immediately'',                                                        !
001500       AT (05,03),                                                      !
001600''or wait until port space is available), or (4) check port for me      !
001700ssage.'',                                                               !
001800       AT (07,03),                                                      !
001900''TYPE:'',                                                              !
002000       AT (07,14), TYPE$, CH(02),                                       !
002100       AT (07,19),                                                      !
002200''(CR-create port; DE-destroy port; XM/XW-transmit; CH-check)'',        !
002300       AT (08,03),                                                      !
002400''PORTNAME:'',                                                          !
002500       AT (08,14), PORTNAME$, CH(04),                                   !
002600       AT (08,19),                                                      !
002700'' '',                                                                  !
002800       AT (09,03),                                                      !
002900''BUFSIZE:'',                                                           !
003000       AT (09,14), BUFSIZE%, PIC(####),                                 !
003100       AT (09,19),                                                      !
003200''(1-2014 bytes - for CR)'',                                            !
003300       AT (10,03),                                                      !
003400''MESSAGE:'',                                                           !
003500       AT (10,14), MESSAGE1$, CH(66),                                   !
003600       AT (11,14), MESSAGE2$, CH(66),                                   !
```

```
003700       AT (12,03),                                                    !
003800''CHECKTYPE:'',                                                       !
003900       AT (12,14), CHECKTYPE$, CH(01),                                !
004000       AT (12,19),                                                    !
004100''(for CH: W-wait;T-interval wait;K-PFkey wait;B-key & interval)'',    !
004200       AT (13,03),                                                    !
004300''INTERVAL:'',                                                        !
004400       AT (13,14), INTERVAL%, PIC(####),                              !
004500       AT (13,19),                                                    !
004600''(for CHECKTYPE=T, time to wait in 1/100 seconds)'',                  !
004700       AT (15,03),                                                    !
004800''RETURN CODE'',                                                      !
004900       AT (15,16), RETURNCODE%   , PIC(##),                           !
005000       AT (19,14),                                                    !
005100''Fill in information and press ENTER for desired action.''
005200RETURN
005300
005400DOMESSAGE:
005410STR(MESSAGE3$,1,66) = MESSAGE1$
005420STR(MESSAGE3$,67,66) = MESSAGE2$
005500   IF TYPE$ = ''CR'' THEN CALL ''MESSAGE'' ADDR(TYPE$,PORTNAME$,      !
005600              BUFSIZE%,RETURNCODE%)
005700   IF TYPE$ = ''DE'' THEN CALL ''MESSAGE'' ADDR(TYPE$,PORTNAME$,      !
005800              RETURNCODE%)
005900   IF TYPE$ = ''XM'' THEN CALL ''MESSAGE'' ADDR(TYPE$,PORTNAME$,      !
006000              MESSAGE3$,132%,RETURNCODE%)
006100   IF TYPE$ = ''CH'' THEN CALL ''MESSAGE'' ADDR(TYPE$,PORTNAME$,      !
006200              CHECKTYPE$,INTERVAL%,MESSAGE3$,132%,RETURNCODE%)
006310MESSAGE1$ = STR(MESSAGE3$,1,66)
006320MESSAGE2$ = STR(MESSAGE3$,67,66)
006400   RETURN
```

MESSAGE Subroutine — A COBOL Example

This program creates a port, transmits a message to the port, retrieves and displays the message, and destroys the port.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MESSAGC.
000300 ENVIRONMENT DIVISION.
000400 DATA OIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION-TYPE PIC X(2) VALUE ''CR''.
000700 77  PORT-NAME PIC X(4) VALUE ''FREO''.
000800 77  THE-MESSAGE PIC X(11) VALUE ''THE MESSAGE''.
000900*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001000*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001100*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001200*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001300*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001400 01  MESSAGE-LENGTH.
001500     03 FILLER USAGE IS BINARY VALUE ZERO.
001600     03 LENGTH-OF-MESSSAGE USAGE IS BINARY VALUE 11.
001700 77  CHECK-TYPE  PIC X(1) VALUE ''W''.
001800 77  RECEIVER PIC X(255) VALUE SPACE.
001900*THE NEXT ITEM MUST BE CODED BUT IS IGNORED SINCE THE CHECK TYPE
002000* IS NOT ''T''.
002100 01 INTERVAL.
002200     03 FILLER USAGE IS BINARY VALUE ZERO.
002300     03 TIME-LENGTH USAGE IS BINARY VALUE ZERO.
002400 01  RECEIVER-LENGTH.
002500     03 FILLER USAGE IS BINARY VALUE ZERO.
002600     03 LENGTH-OF-RECEIVER  USAGE IS BINARY VALUE 255.
002700 01  RETURNCODE.
002800     03 FILLER USAGE IS BINARY VALUE ZERO.
002900     03 ERROR-CODE USAGE IS BINARY.
003000 PROCEDURE DIVISION.
003100 MAIN-PARAGRAPH.
003200     PERFORM CREATE-PARAGRAPH.
003300     PERFORM SEND-PARAGRAPH.
003400     PERFORM CHECK-PARAGRAPH.
003500     PERFORM DESTROY-PARAGRAPH.
003600     STOP RUN.
003700 CREATE-PARAGRAPH.
003800     DISPLAY ''I AM IN THE CREATE-PARAGRAPH''.
003900     CALL ''MESSAGE'' USING FUNCTION-TYPE, PORT-NAME, RETURNCODE.
004000     IF ERROR-CODE NOT EQUAL ZERO DISPLAY ''ERROR-COOE = ''
004100     ERROR-CODE, ELSE DISPLAY ''PORT CREATED''.
```

```
004200 SEND-PARAGRAPH.
004300    DISPLAY ''I AM IN THE SEND-PARAGRAPH.''
004400    MOVE ''XM'' TO FUNCTION-TYPE.
004500    CALL ''MESSAGE'' USING FUNCTION-TYPE, PORT-NAME, THE-MESSAGE,
004600       MESSAGE-LENGTH, RETURNCODE.
004700    IF ERROR-CODE NOT EQUAL ZERO DISPLAY ''ERROR-CODE = ''
004800       ERROR-CODE, ELSE DISPLAY ''MESSAGE DELIVERED''.
004900 CHECK-PARAGRAPH.
005000    DISPLAY ''I AM IN THE CHECK-PARAGRAPH.''
005100    MOVE ''CH'' TO FUNCTION-TYPE.
005200    CALL ''MESSAGE'' USING FUNCTION-TYPE, PORT-NAME, CHECK-TYPE,
005300       INTERVAL, RECEIVER, RECEIVER-LENGTH, RETURNCODE.
005400    IF ERROR-CODE NOT EQUAL ZERO DISPLAY ''ERROR-CODE = ''
005500       ERROR-CODE, ELSE DISPLAY RECEIVER.
005600 DESTROY-PARAGRAPH.
005700    DISPLAY ''I AM IN THE DESTROY-PARAGRAPH.''.
005800    MOVE ''DE'' TO FUNCTION-TYPE.
005900    CALL ''MESSAGE'' USING FUNCTION-TYPE, PORT-NAME, RETURNCODE.
006000    IF ERROR-CODE NOT EQUAL ZERO DISPLAY ''ERROR-CODE = ''
006100       ERROR-CODE, ELSE DISPLAY ''PORT DESTROYED''.
```

MOUNT

FUNCTION

Allows the user to mount a volume.

USAGE (arg1, ..., arg11)

Pos	Argument	Type	Size	Comments
arg1	Device	Integer	4	Device number of the disk or tape to be mounted. Must be nonnegative.
arg2	Volume	Alpha	6	Name of the volume to be mounted.
arg3	Label	Alpha	1	Label type: S or A = Standard label (default) N = No label I = IBM label (tape)
arg4	Mount Usage	Alpha	1	Type of mount: S = Shared (default) E = Exclusive P = Protected (disk) R = Restricted removal (disk)
arg5	Drive Type	Alpha	1	Type of drive (ignored for tape mount): F = Fixed drive R = Removable drive (default)
arg6	System Use Option	Alpha	1	System files that can be written onto the device if the default volume is full (ignored for tape mount): W = Work files S = Spool files A = Work and spool files N = Neither work nor spool files (default)
arg7	Bypass Option	Alpha	1	Bypass label processing option: B = Bypass label Blank = Normal mount (default)
arg8	No-Msg Option	Alpha	1	No mount message option: N = No message (used when the volume is already physically mounted) Blank = Normal mount message (default)
arg9	No-Display Option	Alpha	1	No user display option: N = No display Blank = Normal mount (default) No mount message is displayed at the workstation; a message is usually displayed at the operator console and the user task hangs until the mount is complete.
arg10	Address	Alpha	1	Disk addressing option:

	Option			N = Nonstandard (used for non-Wang soft-sectored diskette) Blank = Standard (default)
arg11	Ret. Code	Integer	4	Error return code. See Table 3-7 below.

NOTES

1. Arguments 3 through 10 are optional; however, if any is included, all preceding arguments must be present. Omitted arguments assume the default values specified in the argument descriptions.

2. All arguments must have acceptable values, even if they are ignored.

Table 3-7. MOUNT Error Return Codes

Return Code	Meaning
0	Successful mount.
4	Successful mount, but new volume label type does not agree with input parameters.
8	Successful mount, but new volume name is not the volume name requested.
12	Disk or tape I/O error detected while reading new volume label or new volume has a bad VTOC. VCBSER set to blank. This return code is set when the new volume is physically mounted on the drive but the VCB cannot be filled in.
16	Device not disk or tape, or device number invalid.
20	Device detached.
24	Disk does not have the requested volume type (fixed or removable).
28	Request to mount an unlabeled volume on a disk unit other than a 2270V diskette.
32	Input volume name blank
36	Requested volume already mounted on a disk unit, or duplicate volume name.
40	Volume currently in use.
44	Currently mounted volume reserved by another user for exclusive use.
48	I/O buffer space insufficient to perform mount.
52	Unable to allocate space for Tape I/O control blocks.
56	Invalid request: work and/or spool filing requested in a nonlabeled volume.
60	Invalid request: nonstandard addressing attempted with Standard Label option or on hard-sectored device.
64	Wrong media: soft-sectored diskette inserted into device for hard-sectored diskettes only.
68	Wrong media: hard-sectored diskette inserted into device for soft-sectored diskettes only.
72	Wrong media: hard-sectored diskette inserted for nonstandard addressing request.
76	Wrong addressing mode: caller requested MOUNT for standard addressing but diskette is nonstandard.
80	Device reserved by another user.
84	MOUNT failed, aborted by user or operator request.

MOUNT Subroutine — A COBOL Example

This program allows the user to mount a nonlabeled diskette.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MOUNTC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000700*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000800*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
000900*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001000*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001100 01  DEVICE.
001200     03 FILLER USAGE IS BINARY VALUE 0.
001300     03 DEVICE-NUMBER USAGE IS BINARY VALUE 23.
001400 77  VOLUME-NAME PIC X(6) VALUE ''FLOPPY''.
001500 77  LABELED PIC X VALUE ''N''.
001600 01  RETURN-KODE.
001700     03 FILLER USAGE IS BINARY VALUE ZERO.
001800     03 ERROR-CODE USAGE IS BINARY.
001900 PROCEDURE DIVISION.
002000 MAIN-PARAGRAPH.
002100     CALL ''MOUNT'' USING DEVICE, VOLUME-NAME, LABELED,
002200       RETURN-KODE.
002300     IF ERROR-CODE = ZERO DISPLAY ''MOUNT SUCCESSFUL''
002400       ELSE DISPLAY ''RETURN CODE = '' ERROR-CODE.
002500     STOP RUN.
```

MOUNT SUBROUTINE - AN RPG II EXAMPLE

This program instructs the MOUNT subroutine to mount a nonlabeled volume called "ARCHIV" on Device 50 of the system. The program checks the return code from the subroutine, displaying it if it is nonzero.

```
00100FDISPLAY DD  F                     WS

00200C                    ACCPTSCR1
00210C*
00220C*         *** PREPARE PARAMETERS TO PASS TO RPGCALL MACRO ***
00230C*
00300C                    Z-ADD50         DEVICE  40
00400C                    MOVE 'ARCHIV'   NAME    6
00500C                    MOVE 'N'        LABEL   1
00600C                    MOVE 'E'        USAGE   1
00700C                    MOVE 'R'        TYPE    1
00800C                    MOVE 'W'        WORK    1
00900C                    MOVE ' '        BYPASS  1
01000C                    MOVE 'N'        NOMESS  1
01200C                    Z-ADD0          RCODE   40
01210C*
01220C*                *** EXIT TO RPGCALL MACRO ***
01230C*
01300C                    EXIT RPGMNT
01400C                    RLABL           DEVICE
01500C                    RLABL           NAME
01600C                    RLABL           LABEL
01700C                    RLABL           USAGE
01800C                    RLABL           TYPE
01900C                    RLABL           WORK
02000C                    RLABL           BYPASS
02100C                    RLABL           NOMESS
02300C                    RLABL           RCODE
02310C*
02320C*               *** CHECK RETURN CODE ***
02330C*
02400C           RCODE    COMP 0                    99
02500C   99               ACCPTSCR3
02600C  N99               ACCPTSCR2
02700C                    SETON                     LR
```

```
02800WSCR1
02900W                  0707        'PRESS ENTER TO MOUNT A'
03000W                  0729        ' NO-LABEL DISKETTE CAL'
03100W                  0751        'LED ''ARCHIV'''
03200W                  0907        'ON SYSTEM DEVICE 50.'
03400WSCR2
03500W                  0707        'MOUNT SUCCESSFUL.  PRE'
03600W                  0729        'SS ENTER TO END JOB.'
03700WSCR3
03800W                  0707        'MOUNT UNSUCCESSFUL;'
03900W                  0907        'RETURN CODE = '
04000W                  0921RCODE
04100W                  1107        'PRESS ENTER TO END JOB'
```

RPGMNT:

```
        RPGCALL   NAME=RPGMNT,CALL=MOUNT,DEVICE,NAME,LABEL,USAGE,TYPE,C
              WORK,BYPASS,NOMESS,(RCODE,4,F)
```

PAUSE

FUNCTION

Causes a program to pause for a user-specified amount of time.

USAGE (arg1)

Pos	Argument	Type	Size	Comments
arg1	Time	Integer	4	Amount of time to pause, in hundredths of a second.

PAUSE Subroutine — A FORTRAN Example

This FORTRAN program first notifies the user that the program is still running, then pauses for one second.

```
C  PROGRAM CODE APPEARS BEFORE THIS STATEMENT
      DO 10 I=1,1000
C  COMPUTATION APPEARS HERE
      IF(I .NE. 500) GO TO 10
      WRITE(0,101)
C
C  CAUSE A ONE SECOND PAUSE
      CALL PAUSE(100)
C
   10 CONTINUE
  101 FORMAT(1X,'COMPUTING')
C  REMAINDER OF PROGRAM FOLLOWS
```

PRINT

FUNCTION

Sends a print file to the print queue.

USAGE (arg1, arg2, ..., arg9)

Pos	Argument	Type	Size	Comments
arg1	File	Alpha	8	Print file submitted by the program.
arg2	Library	Alpha	8	Library on which print file resides. Default is SPOOLIB value, as set with PF2 (SET) of the Command Processor.
arg3	Volume	Alpha	6	Volume on which print file resides. Default is SPOOLVOL value, as set with PF2 (SET) of the Command Processor.
arg4	Mode	Alpha	1	Print mode: S = Spooled (default) H = Hold
arg5	Disposition	Alpha	2	Disposition of file after printing: DS = Dequeue and save (default) DX = Dequeue and scratch RS = Requeue and save
arg6	Copies	Integer	4	Copies to be printed. Default is 1.
arg7	Print Class	Alpha	1	Print class. Must be A-Z or blank. Default is SET PRTCLASS value, as set with PF2 (SET) of the Command Processor.
arg8	Form Number	Integer	4	Form number. Must be 0-255. Default (if omitted or 255) is SET FORM# value, as set with PF2 (SET) of the Command Processor.
arg9	Ret. code	Integer	4	Error return code. See Table 3-8 below.

NOTE

Arguments 2 through 8 are optional. If omitted, defaults are as specified above. If an argument is present, all preceding arguments must also be present.

Table 3-8. PRINT Error Return Codes

Return Code	Meaning
0	Successful.
4	Volume not mounted.
8	Volume in exclusive use.
12	All buffers in use, unable to perform verification.
16	Library not found.
20	File not found.
24	Improper file type, or zero records.
28	File access denied.
32	VTOC error, FDX1 and FDX2 do not agree.
36	VTOC error, FDX2 and FDR do not agree.

PRINT Subroutine — A BASIC Example

This program provides a way of submitting print files that are stored on disk to the printer. The user simply provides the file, library, and volume names. The program displays the default print mode, the disposition of the file after printing, the number of copies, and the form number. The program executes again by flashing the workstation screen briefly and indicating a return code.

```
000100DIM FILE$          08
000200DIM LIBRARY$       08
000300DIM VOLUME$        06
000400DIM MODE$          01
000500DIM DISPOSITION$   02
000600DIM PRINTCLASS$    01
000700MODE$          =''S''
000800DISPOSITION$   =''DS''
000900COPIES%        =0001
001000FORMNUMBER%    =255
001100
001200LOOP:
001300GOSUB PUTSCREEN
001400GOSUB DOPRINT
001500GOTO LOOP
001600
001700PUTSCREEN:
001800ACCEPT                                                              !
001900       AT (01,14),                                                  !
002000''Demonstration of Submit a Print File (PRINT) Subroutine'',        !
002100       AT (03,03),                                                  !
002200''Fill in the following information to submit a print file via the !
002300 PRINT'',                                                           !
002400       AT (04,03),                                                  !
002500''subroutine:'',                                                   !
002600       AT (06,03),                                                  !
002700''FILE:'',                                                         !
002800       AT (06,18), FILE$,         CH(08),                           !
002900       AT (06,29),                                                  !
003000''(Print file to be submitted)'',                                   !
003100       AT (07,03),                                                  !
003200''LIBRARY:'',                                                      !
003300       AT (07,18), LIBRARY$,      CH(08),
003400       AT (08,03),                                                  !
003500''VOLUME:'',                                                       !
003600       AT (08,18), VOLUME$,       CH(06),
003700       AT (09,03),                                                  !
003800''MODE:'',                                                         !
003900       AT (09,18), MODE$,         CH(01),
004000       AT (09,29),                                                  !
004100''(S-spool; H-hold)'',
004200       AT (10,03),                                                  !
004300''DISPOSITION:'',                                                  !
```

```
004400        AT (10,18), DISPOSITION$, CH(02),                                !
004500        AT (10,29),                                                      !
004600''(DS-dequeue & save;DX-dequeue & scratch;RS-requeue)'',                  !
004700        AT (11,03),                                                      !
004800''COPIES:'',                                                             !
004900        AT (11,18), COPIES%,       PIC(####),                            !
005000        AT (12,03),                                                      !
005100''PRINT CLASS:'',                                                        !
005200        AT (12,18), PRINTCLASS$,   CH(01),                               !
005300        AT (13,03),                                                      !
005400''FORM NUMBER:'',                                                        !
005500        AT (13,18), FORMNUMBER%,   PIC(###),                             !
005600        AT (15,03),                                                      !
005700''RETURN CODE:'',                                                        !
005800        AT (15,18), RETURNCODE%,   PIC(##)
005900RETURN
006000
006100DOPRINT:
006200    CALL ''PRINT''  ADDR(FILE$,LIBRARY$,VOLUME$,                         !
006300                MODE$,DISPOSITION$,COPIES%,                              !
006400                PRINTCLASS$,FORMNUMBER%,RETURNCODE%)
006500    RETURN
```

PRINT Subroutine — AN RPG II Example

This program allows the user to print any file in the library #ABCPRT on volume SYSTEM. The user can specify the number of copies desired and whether the file should be scratched after printing.

```
00200FDISPLAY DD  F                     WS

00300C                    ENBLEK0
00400C                    ACCPTSCR1
00410C*
00500C*            *** PREPARE PARAMETERS TO BE PASSED ***
00600C*
01000C                    MOVE '#ABCPRT 'LIBR    8
01100C                    MOVE 'SYSTEM  'VOLM    6
01200C                    MOVE 'S'       MODE    1
01210C             DISP1  COMP 'Y'                          88
01220C  N88                    MOVE 'DS' DISP    2
01300C   88                    MOVE 'DX' DISP    2
01500C                    Z-ADD0         RCODE   40
01510C*
01520C*            *** EXIT TO THE RPGCALL MACRO ***
01530C*
01600C                    EXIT RPGPRT
01700C                    RLABL          FILE
01800C                    RLABL          LIBR
01900C                    RLABL          VOLM
02000C                    RLABL          MODE
02100C                    RLABL          DISP
02200C                    RLABL          COPS
02300C                    RLABL          RCODE
02310C*
02320C*             *** CHECK THE RETURN CODE ***
02330C*
02400C             RCODE  COMP 0                      99
02500C   99               ACCPTSCR2
02600C                    SETON                    LR

02700WSCR1
02800W                0707       'WHICH FILE IN LIBRARY '
02900W                0729       '#ABCPRT WOULD YOU LIKE'
02910W                0752       'TO PRINT?'
03000W                0915                          FILE    8
03100W                1107       'HOW MANY COPIES WOULD '
03200W                1129       'YOU LIKE?'
03300W                1315                          COPS    40
03400W                1507       'SCRATCH THE FILE AFTER'
03410W                1530       'PRINTING? (Y OR N)'
03420W                1715                          DISP1   1
```

```
03500WSCR2
03600W                  0707        'ERROR IN PRINT REQUEST'
03700W                  0729        '; RETURN CODE = '
03800W                  0746RCODE
03900W                  0907        'PRESS ENTER TO END JOB'
```

RPGPRT:

```
         RPGCALL   NAME=RPGPRT,CALL=PRINT,FILE,LIBR,VOLM,MODE,DISP,     C
               (COPS,4,F),(RCODE,4,F)
```

PROTECT

FUNCTION

Changes the protection attributes of a file or library.

USAGE (arg1, ..., arg5, arg6 [repeatable keyword-value pairs], ..., arg8)

Pos	Argument	Type	Size	Comments
arg1	Protect Range	Alpha	1	Indicates scope of protect change: F = Single file L = All files in a library
arg2	File Name	Alpha	8	File whose protect class is to be modified. Must be present, but is ignored if arg1 = L.
arg3	Library	Alpha	8	Library.
arg4	Volume	Alpha	6	Volume.

The following two arguments indicate keyword-value pairs. They can be repeated.

Pos	Argument	Type	Size	Comments
arg5	Keyword	Alpha	2	Specifies the file attribute to change.
arg6	Value	Alpha	var	New value.

Keyword	Recr Type	Recr Size	Receiver Value
ED	Alpha	6	Expiration date, in the form YYMMDD.
FC	Alpha	1	File protection class.
ID	Alpha	3	Owner's ID.

Pos	Argument	Type	Size	Comments
arg7	Limitation Flag	Alpha	1	Access rights: L = Restricted to the user's access rights Blank or omitted = No restriction (use the special access rights of the program, if available) Optional.
arg8	Ret. Code	Integer	4	Error return code. See Table 3-9 below.

NOTE

For FORTRAN programs, the name of this subroutine must be specified as PROTCT.

Table 3-9. PROTECT Error Return Codes

Return Code	Meaning
0	Successful.
4	Volume not mounted.
8	Volume used exclusively by another user.
12	All buffers in use, no protection change.
16	Library not found.
20	File not found.
24	Update access denied, no protection change.
28	(Unused).
32	File in use, no protection change.
36	VTOC error. FDX1 and FDX2 do not agree.
40	VTOC error. FDX2 and FDR do not agree.
44	Invalid argument list address.
48	I/O error. VTOC unreliable.
52	Open or protected files bypassed in protecting library.
56	Invalid new protection data.

PROTECT Subroutine — A BASIC Example

This program allows the user to protect a previously unprotected single file or an entire library on a single volume. The user must also specify the limitation flag, the expiration date, the protect class, and the owner of record.

```
000100DIM RANGE$          01
000200DIM FILE$           08
000300DIM LIBRARY$        08
000400DIM VOLUME$         06
000500DIM LIMIT$          01
000600DIM YY$             02
000700DIM MM$             02
000800DIM DD$             02
000900DIM DATE$           06
001000DIM PROTECTCLASS$ 01
001100DIM OWNER$          03
001200LOOP:
001300GOSUB PUTSCREEN
001400GOSUB PROTECTIT
001500GOTO LOOP
001600
001700PUTSCREEN:
001800ACCEPT                                                              !
001900       AT (01,24),                                                  !
002000''Demonstration of PROTECT Subroutine'',                            !
002100       AT (06,04),                                                  !
002200''Enter the information below to protect a file or a library:'',    !
002300       AT (08,03),                                                  !
002400''RANGE:'',                                                         !
002500       AT (08,22), RANGE$,       CH(01),                            !
002600       AT (08,32),                                                  !
002700''(F-file; L-library)'',                                            !
002800       AT (09,03),                                                  !
002900''FILE:'',                                                          !
003000       AT (09,22), FILE$,        CH(08),                            !
003100       AT (09,32),                                                  !
003200''(ignored if RANGE=L)'',                                           !
003300       AT (10,03),                                                  !
003400''LIBRARY:'',                                                       !
003500       AT (10,22), LIBRARY$,     CH(08),                            !
003600       AT (11,03),                                                  !
003700''VOLUME:'',                                                        !
003800       AT (11,22), VOLUME$,      CH(06),                            !
003900       AT (12,03),                                                  !
004000''LIMITFLAG:'',                                                     !
004100       AT (12,22), LIMIT$,       CH(01),                            !
004200       AT (13,03),                                                  !
004300''EXPIRATION DATE:'',
```

```
004400        AT (13,22), YY$,           CH(02),
004500        AT (13,25), MM$,           CH(02),                                !
004600        AT (13,28), DD$,           CH(02),                                !
004700        AT (13,33),                                                       !
004800''(YY MM DD)'',                                                           !
004900        AT (14,03),                                                       !
005000''PROTECT CLASS:'',                                                       !
005100        AT (14,22), PROTECTCLASS$,CH(01),                                 !
005200        AT (15,03),                                                       !
005300''OWNER OF RECORD:'',                                                     !
005400        AT (15,22), OWNER$,        CH(03),                                !
005500        AT (19,17),                                                       !
005600''Press ENTER to protect either the file or library''
005700RETURN
005800
005900PROTECTIT:
006000DATE$ = YY$ & MM$ & DD$
006100    CALL ''PROTECT'' ADDR(RANGE$,FILE$,LIBRARY$,VOLUME$,                  !
006200        ''ED'',DATE$,                                                     !
006300        ''FC'',PROTECTCLASS$,                                             !
006400        ''ID'',OWNER$,                                                    !
006500        LIMIT$,RETURNCODE%)
006600   PRINT ''RETURN CODE = '': PRINT RETURNCODE%
006700 PRINT ''IF RETURN CODE = 0 SEE IF FILE OR LIBRARY WAS PROTECTED.''
006800 STOP
006900 RETURN
```

PUTPARM

FUNCTION

This subroutine has the following primary functions:

1. Creates a parameter list (called a Parameter Reference Block, or PRB) to satisfy a subsequently generated GETPARM request.

2. Retrieves a previously created PRB.

3. Deletes existing PRBs.

Other functions combine these three.

USAGE (arg1, arguments)

Argument 1 indicates the PUTPARM function and determines the number and nature of other arguments.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Defines the PUTPARM type: D = Create (Display) type E = Create (Enter) type R = Retrieve and Block type M = Retrieve and Merge type C = Cleanup type

Remaining arguments depend on the PUTPARM type selected. A detailed description of each type appears below.

1. Create a Parameter Reference Block (Type = D or E)

This type creates a PRB in one of three ways:

(1) By specifying keywords and values directly;

(2) By referencing a previously created PRB and using its keywords and values to create the new PRB (see Note 1 for a discussion of a limitation of PUTPARM and the use of this feature); and

(3) By referencing a previously created PRB and merging its keywords and values with new keywords and values to create the new PRB.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Value is D or E. D causes a GETPARM screen to be displayed when the PUTPARM call is encountered and allows the user to modify keyword values. E causes no GETPARM interaction; the user-specified PF key indicates the action desired. These types correspond to the DISPLAY and ENTER Procedure language statements.
arg2	Usage Count	Integer	4	Number of times the PRB can be used: 0 = Use generated PRB an unlimited number of times. Other = Use generated PRB arg2 times. Range is 1 to 32768, default is 1. Optional.
arg3	Prname	Alpha	8	Prname of associated GETPARM request to be satisfied. Cannot begin with X'00'.
arg4	Keyword Count	Integer	4	Number of keywords to be associated with this PRB. Arg5-arg7 contain the names and values for these keywords. Range is 0 to 255.
arg5	Keyword	Alpha	8	Name of keyword. Arg5-arg7 are repeated the number of times specified in arg4.
arg6	Value	Alpha	var	Value of keyword.
arg7	Length	Integer	4	Length of value (arg6) in characters. Range is 1 to 256.
arg8	PF Key Value	Alpha	1	Indicates PF key associated with the PRB. If omitted, the default value is "@" (ENTER). See Table 3-18 for AID values.
arg9	PRB Label	Alpha	8	Label to be generated for the PRB. If omitted, or if the label field begins with a blank or X'00', no label is generated.
arg10	Reference Label	Alpha	8	Label of previously defined PRB, whose keywords are to be used in this reference. If this argument begins with a blank or X'00', or is omitted, no "backward reference" is made and the new PRB will be generated directly from arguments 5-7. *Arg4=0:* The program uses all keyword/-value fields in the backward referenced PRB to generate a new PRB. *Arg4#0:* The program creates a PRB consisting of keywords and values specified in args 5-7, updated by values of identical keywords in referenced PRB.
arg11	Cleanup Option	Alpha	1	Indicates action to take after reference to a backward referenced PRB (see arg10): C = Delete backward referenced PRB after reference.

Pos	Argument	Type	Size	Comments
				Blank = Retain backward referenced PRB. Ignored if no backward reference is specified.
arg12	Ret. Code	Integer	4	Error return code. See Table 3-10 below.

Arguments 4 through 11 are optional; however, if any of them is present, all preceding ones must be included (with the exception of Arguments 5 through 7, which must be omitted if arg4=0). Argument 2 can be omitted even if other arguments are specified.

2. Retrieve a previously created parameter reference block (R type)

This type allows the program to examine keywords/values of a previously created PRB, whose values are generally provided for a GETPARM screen request. This option does not create a new PRB; the PRB must have been created earlier in the program (at this level) and must be labeled. This type is generally used to pass file references.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Value is R
arg2	PRB Label	Alpha	8	Label of the previously created PRB that is to be retrieved and examined.
arg3	Receiver	Alpha	var	Receiver for the keyword fields in the previously created PRB. Ignored if arg4=0.
arg4	Receiver Length	Integer	4	Total length of the receiver (arg3). The subroutine returns the PRB fields in the receiver in the order in which they are defined in the PRB, as follows: Byte 1-8 — Keyword 9-12 — Keyword field length 13-end — Keyword field data; length indicated in bytes 9-12 This sequence of bytes repeats until the receiver is filled, or until all keyword fields are transferred. Arg4 is adjusted to reflect the actual number of bytes used.
arg5	Total Length	Integer	4	Total length required by the receiver to hold all keyword field information. If the receiver is large enough to hold all the data, this value will be identical to the value of arg4 on return from the subroutine.
arg6	PF Key Receiver	Alpha	1	Indicates the PF key receiver from the referenced PRB. See Note 4 for a problem with this feature. Optional. See Table 3-18 for AID byte values.
arg7	Cleanup Option	Alpha	1	Indicates the action to take after the PRB has been referenced: C = Delete PRB after fields are extracted Blank = Retain PRB
arg8	Ret. Code	Integer	4	Error return code. See Table 3-10 below.

Arguments 3 through 7 are optional; however, if any of them is present, all preceding arguments must be included. Arguments 3 and 4 must both be either included or omitted.

3. Retrieve a previously created PRB (M type)

The M type allows the program to obtain keyword values from a GETPARM screen. It is generally used to pass file references. The M type is identical in function to the R type; it differs only in the manner in which values in the PRB are returned to the caller.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Value is M
arg2	PRB Label	Alpha	8	Label of the previously created PRB that is to be examined.
arg3	Keyword Count	Integer	4	Number of keywords whose values are to be merged. Each is specified in arguments 4-6.
arg4	Keyword	Alpha	8	Keyword name. Arguments 4-6 are repeated the number of times specified in arg3.
arg5	Receiver	Alpha	var	Receiver for the value of the keyword specified in arg4. If this keyword is found in the "backward referenced" PRB, the receiver contains the value as follows: If the PRB field is longer than the length of arg5, the leftmost arg6 characters are returned. If the PRB field is shorter than arg5, it will be placed left-justified into the field, with the remainder of the field set to blanks.
arg6	Length	Integer	4	Length of the receiver in characters (arg5).
arg7	PF Key Receiver	Alpha	1	Receiver for the PF key value from the referenced PRB. See Note 4 for a problem with this feature. Optional. See Table 3-18 for AID values.
arg8	Cleanup Option	Alpha	1	Indicates action to take after reference to the PRB: C = Delete value after reference Blank = Retain value after reference
arg9	Ret. code	Integer	4	Error return code. See Table 3-10 below.

Arguments 3 through 8 are optional; however, if any of them is present, all preceding arguments must be included.

4. Delete ("cleanup") old parameter reference blocks (C type)

This type causes PRBs created by the program to be removed.

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Value is C
arg2	PRB Label	Alpha	8	Label of PRB to delete. Optional. If omitted, or if the first byte is blank or X'00', all PRBs at this level are deleted.
arg3	Ret. Code	Integer	4	Error return code. See Table 3-10 below.

NOTES

1. Only the PUTPARM user's program can examine or delete a PRB that it has created (via PUTPARM type E or D). This refers specifically to "backward references" (types R and M, and the backward reference option of types E and D) and to "cleanups" (type C and the cleanup option of types E, D, R, and M).

2. A PRB created by the user program can be used to satisfy a GETPARM screen that is exactly one link level beyond it (i.e., in a program linked to, via LINK, by the PUTPARM user's program).

   There are situations in which it is desirable to get around this limitation. For instance, a user menu program might wish to link to another menu which, in turn, runs the COPY utility. The first menu cannot directly create parameters for the COPY screens, since two link levels separate them. However, if the second menu does a PUTPARM type E or D for each of the COPY screens, and specifies a label (arg9) for each of the PRBs, the first menu can create parameters for each of the second menu's PRBs just as if it were a GETPARM screen. The only difference is that the "prname" argument (arg3) in the first menu's PUTPARM should be replaced by the "label" assigned by the second menu. Also, the second menu need not specify any keyword fields in the PRBs, since any fields specified by the first menu are simply added to the second menu's PRBs. The following example helps to clarify this.

   Two BASIC programs might contain the following statements:

   (First menu program, called MENU1)

```
CALL ''PUTPARM'' ADDR (''E'', ''LABLNAME'', 3%, ''FILE     '', FILE$, 8%,
   ''LIBRARY '', LIBRARY$, 8%, ''VOLUME  '', VOLUME$, 6%, RETCODE%)
CALL ''LINK'' ADDR (''MENU2    '', CMPCODE%, RETCODE%)
```

   (Second menu program, called MENU2)

```
CALL ''PUTPARM'' ADDR (''E'', ''INPUT    '', 0%, ''@'', ''LABLNAME'',
    RETCODE%)
CALL ''LINK'' ADDR (''COPY     '', CMPCODE%, RETCODE%)
```

   This example allows the first menu to create parameters for COPY's input screen with FILE$, LIBRARY$, and VOLUME$ even though MENU2 is performing the LINK. LABLNAME is used as the PRB label in MENU2 and as the pseudo-"prname" in MENU1.

   The program can use this method of "chaining" PUTPARMs across as many link levels as desired.

3. The old B and F options have been replaced by the R and M options, respectively. The new options perform the same functions, but their argument lists have been modified. The B and F options still work, but will probably be removed at some point in the future; programs using these options should be modified appropriately.

4. For FORTRAN programs, the name of this subroutine must be specified as PUTPRM.

Table 3-10. PUTPARM Error Return Codes

Return Code	Meaning
0	Successful.
4	Backward referenced label not found.
8	Bad format list supplied.
12	Error found in a previous PRB.
16	Invalid input parameter while using "cleanup" (C) parameter option.
20	Invalid input parameter while using M or R option.

PUTPARM Subroutine — A COBOL Example

This program allows the user to enter file, library, and volume names for corresponding fields of the EDITOR's Input screen. After calling PUTPARM, the program links dynamically to the EDITOR by calling the LINK subroutine. When the EDITOR's Input screen appears, the file, library, and volume fields contain the values entered by the user.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PUTPARMC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600*THE FOLLOWING ITEMS ARE THE ARGUMENTS FOR THE PUTPARM SUBROUTINE
000700 77  TY-PE PIC X VALUE ''D''.
000800 77  PRNAME PIC X(8) VALUE ''INPUT''.
000900 01  KEYWORD-COUNT.
001000     03 FILLER USAGE IS BINARY VALUE 0.
001100     03 WORD-COUNT  USAGE IS BINARY VALUE 4.
001200*THE NEXT TWO ITEMS INITIALIZE THE LANGUAGE FIELD OF THE
001300*EDITOR INPUT SCREEN TO ''C'' FOR COBOL.
001400 77  KEYWORD-1 PIC X(9) VALUE ''LANGUAGE''.
001500 77  LANGUAGE PIC X(9) VALUE ''C''.
001600*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001700*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001800*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001900*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
002000*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
002100 01  LENGTH-1.
002200     03 FILLER USAGE IS BINARY VALUE 0.
002300     03 COUNT-1  USAGE IS BINARY VALUE 9.
002400 77  KEYWORD-2 PIC X(8) VALUE ''FILE''.
002500 77  FILE-NAME PIC X(8) VALUE SPACE.
002600 01  LENGTH-2.
002700     03 FILLER USAGE IS BINARY VALUE 0.
002800     03 COUNT-2  USAGE IS BINARY VALUE 8.
002900 77  KEYWORD-3 PIC X(8) VALUE ''LIBRARY''.
003000 77  LIB-RARY PIC X(8) VALUE SPACE.
003100 01  LENGTH-3.
003200     03 FILLER USAGE IS BINARY VALUE 0.
003300     03 COUNT-3  USAGE IS BINARY VALUE 8.
003400 77  KEYWORD-4 PIC X(8) VALUE ''VOLUME''.
003500 77  VOL-UME PIC X(6) VALUE SPACE.
003600 01  LENGTH-4.
003700     03 FILLER USAGE IS BINARY VALUE 0.
003800     03 COUNT-4  USAGE IS BINARY VALUE 6.
003900 77  PRB-LABEL PIC X(8) VALUE ''EDITPARM''.
004000 01  RETURN-KODE.
004100     03 FILLER USAGE IS BINARY VALUE ZERO.
004200     03 ERROR-CODE  USAGE IS BINARY.
```

```
004300*THE FOLLOWING ITEMS ARE THE ARGUMENTS FOR THE LINK SUBROUTINE
004400 77  LINKNAME  PIC X(8) VALUE ''EDITOR''.
004500 77  LOCATION  PIC X(1) VALUE ''S''.
004600 77  LINK-LIBRARY  PIC X(8).
004700 77  LINK-VOLUME  PIC X(6).
004800 01  PARAMETERS.
004900     03 FILLER USAGE IS BINARY VALUE 0.
005000     03 PARAMETER-COUNT USAGE IS BINARY VALUE 0.
005100 77  EXIT-OPTION  PIC X VALUE ''C''.
005200 77  PF16-MESSAGE   PIC X(18)  VALUE ''RETURN TO PUTPARMC''.
005300 01  MESSAGE-LENGTH.
005400     03 FILLER BINARY VALUE 0.
005500     03 FILLER BINARY VALUE 18.
005600 01  COMPLETION.
005700     03 FILLER USAGE BINARY VALUE ZERO.
005800     03 COMPLETION-CODE USAGE BINARY.
005900 01  ERRORS.
006000     03 FILLER USAGE BINARY VALUE ZERO.
006100     03 LINK-ERROR-CODE USAGE BINARY VALUE ZERO.
006200 PROCEDURE DIVISION.
006300 MAIN-PARAGRAPH.
006400     ACCEPT FILE-NAME, LIB-RARY, VOL-UME.
006500     CALL ''PUTPARM'' USING TY-PE, PRNAME, KEYWORD-COUNT,
006600        KEYWORD-1, LANGUAGE, LENGTH-1, KEYWORD-2, FILE-NAME,
006700        LENGTH-2, KEYWORD-3, LIB-RARY, LENGTH-3, KEYWORD-4,
006800        VOL-UME, LENGTH-4, RETURN-KODE.
006900     IF ERROR-CODE NOT EQUAL ZERO, GO TO PUTPARM-ERROR.
007000     CALL ''LINK'' USING LINKNAME, LOCATION, LINK-LIBRARY,
007100        LINK-VOLUME, PARAMETERS, EXIT-OPTION,
007200        PF16-MESSAGE, MESSAGE-LENGTH,
007300        COMPLETION, ERRORS.
007400     IF COMPLETION-CODE = 8 DISPLAY ''LINK-ERROR-CODE IS ''
007500        LINK-ERROR-CODE, ELSE DISPLAY ''YAY!''.
007600     STOP RUN.
007700 PUTPARM-ERROR.
007800     DISPLAY ''PUTPARM ERROR-CODE = '' ERROR-CODE.
007900     STOP RUN.
```

PUTPARM Subroutine — AN RPG II Example

This program calls the PUTPARM subroutine four times. Each time, PUTPARM is used to supply parameters for one of the GETPARM screens displayed by the COPY utility. The program then calls the LINK subroutine to link to the COPY utility. In the COPY utility, the file EXPENSES in library ABCDATA on volume SYSTEM is copied to a file called EXPENSE2 in the same library. The user does not see any of COPY's GETPARM screens, since the PUTPARM type is E (Enter) rather than D (Display).

```
00100FDISPLAY DD  F                     WS
00200C                    ACCPTSCR1
00210C*
00220C*     *** PREPARE PARAMETERS TO PASS DURING FIRST PUTPARM CALL ***
00230C*
00300C                    MOVE 'E'        TYPE    1
00400C                    MOVE 'INPUT   'PRN      8
00500C                    Z-ADD4          KCNT    40
00600C                    MOVE 'FILE    'KEY1     8
00700C                    MOVE 'EXPENSES'VAL1     8
00800C                    Z-ADD8          LEN1    40
00900C                    MOVE 'LIBRARY 'KEY2     8
01000C                    MOVE 'ABCDATA' VAL2     7
01100C                    Z-ADD7          LEN2    40
01200C                    MOVE 'VOLUME  'KEY3     8
01300C                    MOVE 'SYSTEM'  VAL3     6
01400C                    Z-ADD6          LEN3    40
01500C                    MOVEL'COPY    'KEY4     8
01600C                    MOVE 'FILE'    VAL4     4
01700C                    Z-ADD4          LEN4    40
01800C                    Z-ADD0          RCOD    40
01810C*
01820C*       *** EXIT TO RPGCALL MACRO ***
01830C*
01900C                    EXIT RPGPTA
02000C                    RLABL           TYPE
02100C                    RLABL           PRN
02200C                    RLABL           KCNT
02300C                    RLABL           KEY1
02400C                    RLABL           VAL1
02500C                    RLABL           LEN1
02600C                    RLABL           KEY2
02700C                    RLABL           VAL2
02800C                    RLABL           LEN2
02900C                    RLABL           KEY3
03000C                    RLABL           VAL3
03100C                    RLABL           LEN3
03200C                    RLABL           KEY4
03300C                    RLABL           VAL4
03400C                    RLABL           LEN4
03500C                    RLABL           RCOD
```

```
03510C*
03520C*       ***  CHECK RETURN CODE ***
03530C*
03600C           RCOD      COMP 0                     10
03700C   10                GOTO ERRS
03710C*
03720C*    *** PREPARE PARAMETERS TO PASS DURING SECOND PUTPARM CALL ***
03730C*
03800C                     MOVE 'OPTIONS 'PRN
03900C                     Z-ADD0         KCNT
04000C                     Z-ADD0         RCOD
04010C*
04020C*                 *** EXIT TO RPGCALL MACRO ***
04030C*
04100C                     EXIT RPGPTB
04510C*
04520C*                   *** CHECK RETURN CODE ***
04530C*
04600C           RCOD      COMP 0                     20
04700C   20                GOTO ERRS
04710C*
04720C*     *** PREPARE PARAMETERS TO PASS DURING THIRD PUTPARM CALL ***
04730C*
04800C                     MOVE 'OUTPUT  'PRN
04900C                     Z-ADD3         KCNT
05000C                     MOVE 'EXPENSE2'VAL1
05100C                     Z-ADD0         RCOD
05101C*
05110C*       *** EXIT TO RPGCALL MACRO ***
05120C*
05200C                     EXIT RPGPTC
06150C*
06155C*                 *** CHECK RETURN CODE ***
06160C*
06200C           RCOD      COMP 0                     30
06300C   30                GOTO ERRS
06310C*
06320C*  *** PREPARE PARAMETERS TO PASS DURING FOURTH PUTPARM CALL ***
06330C*
06400C                     MOVE 'EOJ     'PRN
06500C                     Z-ADD0         KCNT
06600C                     MOVE 'P'       PFK         1
06700C                     Z-ADD0         RCOD
06710C*
06720C*                *** EXIT TO RPGCALL MACRO ***
06730C*
06800C                     EXIT RPGPTD
06900C                     RLABL          PFK
06920C*
06930C*                 *** CHECK RETURN CODE ***
06940C*
07000C                RCOD   COMP 0                     40
07100C   40                GOTO ERRS
```

```
07101C*
07110C*     *** PREPARE PARAMETERS TO PASS DURING LINK CALL ***
07120C*
07200C                    MOVE 'COPY    'PROG    8
07300C                    MOVE 'S'      LTYPE    1
07400C                    Z-ADD0        CCODE    40
07500C                    Z-ADD0        RCODE    40
07510C*
07520C*              *** EXIT TO RPGCALL MACRO ***
07530C*
07600C                    EXIT RPGPTE
07700C                    RLABL         PROG
07800C                    RLABL         LTYPE
07900C                    RLABL         CCODE
08000C                    RLABL         RCODE
08010C*
08020C*          *** CHECK COMPLETION AND RETURN CODES ***
08030C*
08100C           CCODE    COMP 0                   50
08200C           RCODE    COMP 0                   50
08300C  N50               GOTO END
08310C*
08320C*          *** DISPLAY ERROR MESSAGE SCREEN ***
08330C*
08400C           ERRS     TAG
08500C                    ACCPTSCR2
08600C           END      TAG
08700C                    SETON                    LR
08800WSCR1
08900W           1207     'PRESS ENTER TO COPY TH'
09000W           1229     'E EXPENSES FILE INTO A'
09100W           1251     ' FILE CALLED EXPENSE2.'
09200WSCR2
09300W      N50  0707     'ERROR IN PUTPARM CALL '
09400W       10  0729     'DEFINING INPUT.'
09500W       20  0729     'DEFINING OPTIONS.'
09600W       30  0729     'DEFINING OUTPUT.'
09700W       40  0729     'DEFINING EOJ.'
09800W      N50  1010     'RETURN CODE = '
09900W      N50  1025RCOD
10000W       50  0707     'ERROR IN LINK CALL.'
10100W       50  1010     'COMPLETION CODE = '
10200W       50  1030CCODE
10300W       50  1210     'RETURN CODE = '
10400W       50  1225RCODE
```

RPGPTA:

```
RPGCALL   NAME=RPGPTA,CALL=PUTPARM,TYPE,PRN,(KCNT,4,F),          C
      KEY1,VAL1,(LEN1,4,F),KEY2,VAL2,(LEN2,4,F),KEY3,VAL3,
      (LEN3,4,F),KEY4,VAL4,(LEN4,4,F),(RCOD,4,F)                 C
```

RPGPTB:

```
RPGCALL   NAME=RPGPTB,CALL=PUTPARM,TYPE,PRN,(KCNT,4,F),          C
      (RCOD,4,F)
```

RPGPTC:

```
RPGCALL   NAME=RPGPTC,CALL=PUTPARM,TYPE,PRN,(KCNT,4,F),KEY1,     C
      VAL1,(LEN1,4,F),KEY2,VAL2,(LEN2,4,F),KEY3,VAL3,
      (LEN3,4,F),(RCOD,4,F)                                      C
```

RPGPTD:

```
RPGCALL   NAME=RPGPTD,CALL=PUTPARM,TYPE,PRN,(KCNT,4,F),PFK,      C
      (RCOD,4,F)
```

RPGPTE:

```
RPGCALL   NAME=RPGPTE,CALL=LINK,PROG,LTYPE,(CCODE,4,F),          C
      (RCODE,4,F)
```

READFDR

FUNCTION

Obtains information about a specified file. READFDR can return specified control blocks or various characteristics about the file. The control blocks and characteristics appear below.

USAGE (arg1, ..., arg4, arguments)

Arg1 through arg3 identify the file about which information is obtained. Arg4 defines the function to be performed and the number and nature of the additional arguments.

Pos	Argument	Type	Size	Comments
arg1	File	Alpha	8	File whose FDR(s) and/or AXD1 are to be retrieved.
arg2	Library	Alpha	8	Library containing the file.
arg3	Volume	Alpha	6	Volume being accessed.
arg4	Function	Integer	4	Type of information to be returned: 0 = Return specified control blocks 1 = Return FDR1 2 = Return FDR2 3 = Return FDR1 and FDR2 4 = Return AXD1 5 = Return FDR1 and AXD1 6 = Return FDR2 and AXD1 7 = Return FDR1 and FDR2 and AXD1

The remaining arguments depend on the function type.

1. Return specified control blocks (arg4 is nonzero)

Pos	Argument	Type	Size	Comments
arg5	Receiver	Alpha	var	Data item that receives the blocks specified in arg4. Its length depends on which blocks are returned. FDR1 and FDR2 are 80 bytes each. AXD1 is 60 bytes plus 28 bytes for each alternate key. The maximum length for AXD1 is 2048 bytes. The order in which the blocks are received is specified in arg4.
arg6	Ret. Code	Integer	4	Error return code. See Table 3-11 below. If the return code is nonzero, only FDR1 and FDR2 are returned.

2. Return specified fields (arg4 is zero)

Each of the keywords is Alpha(2). Definitions of the type and contents of the receivers appear in the following list. The last argument in the argument list (arg5) must be the error return code.

Keyword	Recr Type	Recr Size	Receiver Value
AC	Integer	4	Number of defined alternate keys.
AX	Alpha	var	Alternate key information entry. Must be at least 12 times the number of alternate keys. This information is not available if the return code (arg5) is nonzero. Each entry is 12 bytes and consists of the following: Byte 1-2 — Alternate key number. 3-4 — Position of the key field in record. 5-6 — Key length. 7 — Duplicates flag: D = Duplicate alternate keys allowed Blank = Duplicates not allowed 8 — Compression flag: C = Key entries are compressed Blank = Key entries not compressed 9-12 — Number of records on this alternate key path.
BA	Integer	4	Number of blocks allocated to the file.
BC	Integer	4	Number of blocks used by the file.
CD	Alpha	6	Creation date of the file, in the form YYMMDD.
DP	Integer	4	Data packing factor.
EA	Integer	4	Number of extents allocated to the file.
ED	Alpha	6	Expiration date of the file, in the form YYMMDD.
EL	Alpha	var	Starting and ending sectors of the extents allocated to the file, listed in pairs of 4-byte integer entries. The length of the EL receiver must be at least eight times the value of the EA receiver.
FC	Alpha	1	File protection class.
FT	Alpha	1	File type: C = Consecutive I = Indexed P = Print O = Object program A = Alternate indexed L = Log W = Word processing document

Keyword	Recr Type	Recr Size	Receiver Value
ID	Alpha	3	File creator's ID.
IP	Integer	4	Index packing factor.
KP	Integer	4	Position of the first byte of the primary key (counting from 1).
KS/KL	Integer	4	Length of the primary key.
MD	Alpha	6	Date of the last modification to the file, in the form YYMMDD.
ME	Alpha	4	Special execute access flags for the file.
MR	Alpha	4	Special read access flags for the file.
MW	Alpha	4	Special write access flags for the file.
PF	Alpha	1	Partial file indicator, created by BACKUP utility: P = Partial file Blank = Normal file
RC	Integer	4	Number of records in the file.
RS	Integer	4	Size of the records in the file. For fixed length records, this is the actual size. For variable length records, this is the specified maximum size.
RT	Alpha	1	Record type: F = Fixed-length V = Variable-length C = Compressed

Pos	Argument	Type	Size	Comments
arg5	Ret. Code	Integer	4	Error return code. Code 100, 104, or 108 returned only for an unsuccessful attempt to access AXD1 and only if no error has occurred in attempting to access FDR1 and FDR2. See Table 3-11 below.

NOTE

For FORTRAN programs, the name of this subroutine must be specified as RDFDR.

Table 3-11. READFDR Error Return Codes

Return Code	Meaning
0	Operation performed successfully.
4	Volume not mounted.
8	Volume used exclusively by another user.
12	All buffers in use.
16	Library not found.
20	File label not found.
32	VTOC error. FDX1 and FDX2 do not agree.
36	VTOC error. FDX2 and FDR do not agree.
40	Invalid input parameters.
44	Disk I/O error. VTOC unreliable.
100	Possession conflict.
104	Protection violation.
108	Partial BACKUP file (cannot be opened).

READFDR Subroutine — A COBOL Example

This program accepts file, library, and volume names specified by the user. It also returns the number of blocks allocated for the file, the number of blocks used, the number of extents allocated, and the file's data packing factor.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. READFDRC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FILE-NAME  PIC X(8).
000700 77  LIB-RARY  PIC X(8).
000800 77  VOL-UME  PIC X(6).
000900*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001000*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001100*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001200*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001300*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001400 01  FUNCTION.
001500     03 FILLER USAGE IS BINARY VALUE ZERO.
001600     03 FUNCTION-CODE USAGE IS BINARY VALUE 0.
001700 77  BLOCKS-ALLOCATED PIC X(2) VALUE ''BA''.
001800 01  NUMBER-ALLOCATED.
001900     03 FILLER USAGE IS BINARY VALUE ZERO.
002000     03 ALLOCATED USAGE IS BINARY.
002100 77  BLOCKS-USED PIC X(2) VALUE ''BC''.
002200 01  NUMBER-USED.
002300     03 FILLER USAGE IS BINARY VALUE ZERO.
002400     03 USED USAGE IS BINARY.
002500 77  EXTENT-KEY PIC X(2) VALUE ''EA''.
002600 01  EXTENT-NUMBER.
002700     03 FILLER USAGE IS BINARY VALUE ZERO.
002800     03 EXTENTS USAGE IS BINARY.
002900 77  DATA-PACK-KEY PIC X(2) VALUE ''DP''.
003000 01  DATA-PACK-NUMBER.
003100     03 FILLER USAGE IS BINARY VALUE ZERO.
003200     03 DATA-PACK  USAGE IS BINARY.
003300 01  RETURNCODE.
003400     03 FILLER USAGE IS BINARY VALUE ZERO.
003500     03 ERROR-CODE USAGE IS BINARY.
```

```
003600 PROCEDURE DIVISION.
003700 FIRST-PARAGRAPH.
003800     ACCEPT FILE-NAME, LIB-RARY, VOL-UME.
003900     IF FILE-NAME = ''ABC'' GO TO EXIT-PARAGRAPH.
004000     CALL ''READFDR'' USING FILE-NAME, LIB-RARY, VOL-UME, FUNCTION,
004100          BLOCKS-ALLOCATED, NUMBER-ALLOCATED,
004200          BLOCKS-USED, NUMBER-USED, EXTENT-KEY, EXTENT-NUMBER,
004300          DATA-PACK-KEY, DATA-PACK-NUMBER, RETURNCODE.
004400     IF ERROR-CODE NOT = 0 DISPLAY ''RETURN CODE = ''ERROR-CODE,
004500          GO TO EXIT-PARAGRAPH.
004600     DISPLAY ''ALLOCATED = ''ALLOCATED,
004700             '' USED = ''USED,
004800             '' EXTENTS = ''EXTENTS,
004900             '' DATA-PACK = ''DATA-PACK.
005000     GO TO FIRST-PARAGRAPH.
005100 EXIT-PARAGRAPH.
005200     STOP RUN.
```

READVTOC

FUNCTION

Provides information from the Volume Table of Contents (VTOC). The available information includes the following:

Files in a specified library
Libraries on a specified volume
Standard label volumes on the system
Free extents on a volume
General information about a volume
Specified VTOC blocks
Files and free extents on a volume

USAGE (arg1, arguments)

Arg1 defines the function to be performed and the number and nature of the additional arguments.

1. Obtain the names of files in a specified library

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is F
arg2	Library	Alpha	8	Library containing the files.
arg3	Volume	Alpha	6	Volume being accessed.
arg4	Starter	Integer	4	File entry at which to begin listing. Must be nonnegative. Value of 0 is treated as 1.
arg5	Counter	Integer	4	Number of file entries to list. Must be nonnegative. If fewer entries are returned than specified, arg5 is set to the actual number of entries returned.
arg6	Receiver	Alpha	var	Data item that receives the file entries. The length must be at least eight times the value of arg5. Each entry is 8 bytes and contains one file name.
arg7	Ret. Code	Integer	4	Error return code. See Table 3-12 below.
arg8	File Count	Integer	4	Number of files in the specified library.

2. Obtain the names of libraries on a specified volume

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is L
arg2	Volume	Alpha	6	Volume containing the libraries.
arg3	Starter	Integer	4	Library entry at which to begin listing. Must be nonnegative. A value of 0 is treated as 1.
arg4	Counter	Integer	4	Number of library entries to list. Must be nonnegative. If fewer entries are returned than specified, arg4 is set to the actual number of entries returned.
arg5	Receiver	Alpha	var	Data item that receives the library entries. The length must be at least 10 times the value of arg4. Each library entry is 10 bytes and contains the library name (the first 8 bytes) and the number of files in the library (the last 2 bytes).
arg6	Ret. Code	Integer	4	Error return code. See Table 3-12 below.
arg7	Library Count	Integer	4	Number of libraries on the specified volume.

3. Obtain the names of standard label (SL) volumes on the system

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is V
arg2	Starter	Integer	4	Volume entry at which to begin listing. Must be nonnegative. A value of 0 is treated as 1.
arg3	Counter	Integer	4	Number of volume entries to list. Must be nonnegative. If fewer entries are returned than specified, arg3 is set to the actual number of entries returned.
arg4	Receiver	Alpha	var	Data item that receives the volume entries. The length must be at least 16 times the value of arg3. Each item is 16 bytes and is structured as follows: Byte 1-6—Volume name. 7-8—X'00' (unused). 9-10—Total number of libraries. 11-12—Total number of files. 13-16—Error return code. See Table 3-12 below. If the return code is nonzero, Bytes 9-12 are not set.
arg5	Volume	Integer	4	Number of SL disk volumes on the system.

4. Obtain the locations of free extents on a volume

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is X
arg2	Volume	Alpha	6	Volume whose free extents are to be listed.
arg3	Starter	Integer	4	Relative order of the entry at which to begin the listing. A value of 0 is treated as 1.
arg4	Counter	Integer	4	Number of extent entries to list. Must be non-negative. If fewer entries are returned than specified, arg4 is set to the actual number of entries returned.
arg5	Receiver	Alpha	var	Data item that receives the extent entries. Must be at least eight times the value of arg4. Each entry consists of two four-byte integers containing the starting and ending block numbers for the extent.
arg6	Ret. Code	Integer	4	Error return code. See Table 3-12 below.
arg7	Extents	Integer	4	Number of free extents on the specified volume.

5. Obtain general information about a volume

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is G
arg2	Volume	Alpha	6	Volume whose VTOC is being read.
arg3	Keyword	Alpha	2	Type of information to be returned in arg4. Each arg3 must be paired with arg4 and can be repeated.
arg4	Receiver	Integer	4	Receives the information specified by arg3. Each arg4 must be paired with arg3. The keywords and information received are as follows.

Keyword	Contents of receiver
BC	Number of blocks on the volume available to the user.
BF	Number of free user blocks on the volume.
DC	Number of blocks on the volume, not including the spare cylinder.
FC	Number of files on the volume.
LC	Number of libraries on the volume.
PC	Number of physical blocks on the volume, including the spare cylinder.
VC	Number of blocks in the VTOC.

Keyword	Contents of receiver
VF	Number of free blocks in the VTOC. The maximum value returned is 255; therefore, for large disks, this result may be meaningless. (A value of exactly 255 can probably be dismissed as incorrect.)
XF	Number of free user extents on the volume.

Pos	Argument	Type	Size	Comments
arg5	Ret. Code	Integer	4	Error return code. See Table 3-12 below.

6. Obtain specified VTOC blocks

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is #
arg2	Volume	Alpha	6	Volume whose VTOC is being read.
arg3	Starter	Integer	4	Entry at which to begin listing. Must be non-negative. A value of 0 is treated as 1.
arg4	Counter	Integer	4	Number of VTOC blocks to return. Must be nonnegative. If fewer blocks are returned, arg5 is set to the number of blocks returned.
arg5	Receiver	Alpha	var	User file UFB (file number in BASIC, or FD in COBOL) which receives the VTOC blocks requested. The file must consist of 2048-byte records and must be open in Output mode; there should be as least as much space in the file as specified by arg4. Any existing records in the file are destroyed.
arg6	Ret. Code	Integer	4	Error return code. See Table 3-12 below.
arg7	Blocks	Integer	4	Number of blocks in the VTOC.

7. Obtain the names of files and the locations of free extents on a volume

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is D
arg2	Volume	Alpha	6	Volume whose VTOC is to be read.
arg3	File Starter	Integer	4	Relative order of VTOC entry at which to begin listing. Must be nonnegative. Value of 0 is treated as 1.
arg4	File Counter	Integer	4	Number of file names to return. Must be non-negative. If fewer entries are returned, arg4 is set to the number of entries returned.

Pos	Argument	Type	Size	Comments
arg5	File Receiver	Alpha	var	FD name (COBOL) or file number (BASIC) of a file that receives the entries for the files on the volume. This file must be open in either Output or Extend mode; on return from the subroutine, it is open in Extend mode. The records in the file must consist of 182-byte consecutive records and will have the following structure: Byte 1-6—Volume name 7-14—Library name 15-22—File name 23-102—FDR1 record for the file 103-182—FDR2 record for the file, or zeroes if none
arg6	Extent Starter	Integer	4	Free extent at which to begin listing. Optional.
arg7	Extent Counter	Integer	4	Number of free extent entries to return. Optional. If fewer entries are returned than specified, arg6 is set to the actual number of entries returned.
arg8	Extent Receiver	Alpha	var	FD name (COBOL) or number (BASIC) of the file that receives the entries. Optional. This file must be open in Output or Extend mode and consists of 8-byte consecutive records. Bytes 1 to 4 contain the starting block number of a free extent; bytes 5 to 8 contain the ending block number. Upon return from the subroutine, this file is open in the Extend mode.
arg9	Ret. Code	Integer	4	Error return code. See Table 3-12 below.
arg10	Files	Integer	4	Number of files on the volume, computed from the VTOC blocks.
arg11	Extents	Integer	4	Number of free extents on the volume, computed from the VTOC blocks. Optional. See Note 1.

Arguments 6 through 8 and 11 must all be either present or absent. These options can be used for VTOC verification, since the free extent information and the file information are extracted from the same VTOC state. If verification is not desired or if the VTOC is guaranteed to be unchanging, the programmer can use the X function (function 4) to retrieve the same free extent information without requiring a user file for the output.

NOTES

1. This subroutine makes two important assumptions:

   (a) That the disk volume has a readable VTOC; otherwise, the results are not predictable and the user file records and/or free extent records might contain incorrect information.

(b) That the current structure of the VS VTOC is the basis for the subroutine. Should this change in a future release, a new version of the subroutine would be required to ensure correct processing of this option.

2. For FORTRAN programs, the name of this subroutine must be specified as RDVTOC.

3. The General Information option (G) replaces the Extends option (B), which continues to be supported.

Table 3-12. READVTOC Error Return Codes

Return Code	Meaning
0	Successful.
4	Invalid argument list address.
8	Volume not mounted.
12	Volume used exclusively by another user.
16	All buffers in use.
20	Volume specified is nonlabeled.

READVTOC Subroutine — A COBOL Example

This program retrieves the names of the files in a library specified by the user. Ten files are read simultaneously. The program also returns the number of files in the library. Output appears on the workstation.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. RDVTOCC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  TY-PE PIC X VALUE ''F''.
000700 77  LIB-RARY PIC X(8).
000800 77  VOL-UME PIC X(6).
000900*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001000*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001100*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001200*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001300*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001400 01  STARTER.
001500     03 FILLER USAGE IS BINARY VALUE ZERO.
001600     03 STARTNUMBER  USAGE IS BINARY.
001700 01  COUN-TER.
001800     03 FILLER USAGE IS BINARY VALUE 0.
001900     03 COUNTNUMBER USAGE IS BINARY.
002000 77  RECEIVER PIC X(80).
002100 01  RETURNCODE.
002200     03 FILLER USAGE IS BINARY VALUE ZERO.
002300     03 RETURNVALUE USAGE IS BINARY.
002400 01  FILE-COUNT.
002500     03 FILLER USAGE IS BINARY VALUE ZERO.
002600     03 FILECOUNT USAGE IS BINARY.
002700 PROCEDURE DIVISION.
002800 MAIN-PARAGRAPH.
002900     ACCEPT LIB-RARY, VOL-UME.
003000     IF LIB-RARY = ''X'' GO TO STOP-PARAGRAPH.
003100*COUNTNUMBER MUST BE INITIALIZED WHENEVER A NEW LIBRARY IS READ,
003200*SINCE THE VALUE RETURNED MAY THE LESS THAN THE ORIGINAL.
003300     MOVE 10 TO COUNTNUMBER.
003400     PERFORM CALL-PARAGRAPH VARYING STARTNUMBER FROM 1 BY 10
003500          UNTIL COUNTNUMBER LESS THAN 10.
003600     GO TO STOP-PARAGRAPH.
003700 CALL-PARAGRAPH.
003800     MOVE SPACES TO RECEIVER.
003900     CALL ''READVTOC'' USING TY-PE, LIB-RARY, VOL-UME, STARTER,
004000          COUN-TER, RECEIVER, RETURNCODE, FILE-COUNT.
004100     IF RETURNVALUE NOT = 0 DISPLAY ''RETURN CODE = ''RETURNVALUE,
004200          GO TO STOP-PARAGRAPH.
004300     DISPLAY RECEIVER.
004400     IF STARTNUMBER = 1 DISPLAY ''FILECOUNT = ''FILECOUNT.
004500     DISPLAY ''COUNTNUMBER = ''COUNTNUMBER.
004600 STOP-PARAGRAPH.
004700     STOP RUN.
```

RENAME

FUNCTION

Allows the user to rename a file or library, with the options of bypassing expiration date checking and limiting access rights for a program with special privileges.

USAGE (arg1, ..., arg10)

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Type of rename: Specify F to rename a file, L to rename a library, G to rename a file across library boundaries (specify new file and library names).
arg2	File Name	Alpha	8	Name of the file to be renamed. Ignored if arg1=L.
arg3	Library	Alpha	8	Rename library: *Arg1=F:* Library where file resides. *Arg1=L or G:* Library to be renamed.
arg4	Volume	Alpha	6	Volume where library resides.
arg5	New File	Alpha	8	New file name.
arg6	New Lib.	Alpha	8	New library name. Optional, but required for library rename.
arg7	Bypass Flag	Alpha	1	Indicates whether to bypass expiration date checking. Optional. B = Bypass checking Blank = Do not bypass checking.
arg8	Access Limit Flag	Alpha	1	Access rights to the new file or library: L = Restrict rights to the access rights of the program user. Blank = Allow full access privileges. Optional.
arg9	Allow-OPEN Flag	Alpha	1	Rename-when-open option: O = Allow rename when open Blank = Do not allow rename Optional.
arg10	Ret. Code	Integer	4	Error return code. See Table 3-13 below.

NOTES

1. The user cannot rename a library that contains a file for which the user does not have update access rights.

2. Any arguments that are omitted have the default values associated with the user.

3. Arguments 6 through 9 are optional, but if any of them is present, all preceding arguments must also be present.

Table 3-13. RENAME Error Return Codes

Return Code	Meaning
0	File or library renamed.
4	Volume not mounted.
8	Volume used exclusively by other user.
12	All buffers in use, no rename.
16	Library not found.
20	File not found.
24	Update access to some file protection class in the library denied, no rename.
28	Unexpired file, no rename.
32	File in use, no rename.
36	VTOC error. FDX1 and FDX2 do not agree.
40	VTOC error. FDX2 and FDR do not agree.
44	Invalid argument list address.
48	I/O error. VTOC unreliable.
52	New file name or library name already exists, no rename.
56	New file name invalid, or first character is #, no rename.
60	VTOC currently full. Insufficient space for new FDX1/FDX2.
64	Reserved bits in parameter list options byte are nonzero.

RENAME Subroutine — A COBOL Example

This program allows the user to change the name of a file whose retention period might not have expired. Argument 7 is omitted because access rights are not restricted.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. RENAMEC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION  PIC X VALUE ''G''.
000700 77  FILE-NAME  PIC X(8).
000800 77  LIB-RARY  PIC X(8).
000900 77  VOL-UME  PIC X(6).
001000 77  NEW-NAME  PIC X(8).
001100 77  EXPIRE-CHECK PIC X VALUE ''B''.
001200*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001300*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001400*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001500*BYTES FOR THE INTEGER.
001600 01  RETURNCODE.
001700     03 FILLER USAGE IS BINARY VALUE ZERO.
001800     03 ERROR-CODE USAGE IS BINARY.
002900 PROCEDURE DIVISION.
002000 FIRST-PARAGRAPH.
002100     ACCEPT FILE-NAME, LIB-RARY, VOL-UME, NEW-NAME.
002200     IF FILE-NAME = ''ABC'' GO TO EXIT-PARAGRAPH.
002300     PERFORM CALL-PARAGRAPH.
002400 CALL-PARAGRAPH.
002500     CALL ''RENAME'' USING FUNCTION, FILE-NAME, LIB-RARY, VOL-UME,
002600         NEW-NAME, EXPIRE-CHECK, RETURNCODE.
002700     IF ERROR-CODE NOT EQUAL ZERO GO TO ERROR-PARAGRAPH.
002800     DISPLAY ''TO VERIFY USE PF KEY 5 FROM THE COMMAND PROCESSOR.''
002900     GO TO FIRST-PARAGRAPH.
003000 ERROR-PARAGRAPH.
003100     DISPLAY ''ERROR-CODE = ''ERROR-CODE.
003200     GO TO FIRST-PARAGRAPH.
003300 EXIT-PARAGRAPH.
003400     STOP RUN.
```

RENAME Subroutine — A FORTRAN Example

This example is a general purpose, interactive program that allows the user to rename a file or library by providing names during program execution.

```
      LOGICAL*1 TYPE, EXP, LIM
      REAL*8 FILE, LIB, VOL, NEW
C  RCODE IS THE RETURN CODE FOR THE SUBROUTINE
      INTEGER RCODE
C  ASK THE USER FOR THE NECESSARY INPUTS
      PRINT 101,' RENAME FILE (F) OR LIBRARY (L)?'
      READ(0,103) TYPE
      IF(TYPE .EQ. 'L')GO TO 10
      PRINT 101,' ENTER NAME OF FILE TO BE RENAMED'
      READ(0,102) FILE
      PRINT 101,' ENTER NAME OF LIBRARY'
      READ(0,102) LIB
      GO TO 20
   10 PRINT 101,' ENTER LIBRARY TO BE RENAMED'
      READ(0,102) LIB
   20 PRINT 101,' ENTER VOLUME NAME'
      READ(0,102) VOL
      PRINT 101,' ENTER NEW FILE/LIBRARY NAME'
      READ(0,102) NEW
C  SET EXPIRATION DATE AND ACCESS LIMITS
      EXP = 'B'
      LIM = 'L'
C
C  CALL THE RENAME SUBROUTINE
      CALL RENAME (TYPE, FILE, LIB, VOL, NEW, EXP, LIM, RCODE)
C
C  PRINT THE RETURN CODE
      PRINT 104, RCODE
  101 FORMAT(A35)
  102 FORMAT(A8)
  103 FORMAT(A1)
  104 FORMAT(1X, 'RETURN CODE = 'I4)
      PAUSE
      END
```

WANG

Customer Comment Form

Title VS USERSUBS REFERENCE

Publications Number 800-1315US-01

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other _____

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ **Street** _____

Title _____ **City** _____

Dept/Mail Stop _____ **State/Country** _____

Company _____ **Zip Code** _____ **Telephone** _____

Thank you for your help.

Printed in U.S.A. 14-3140 3-82-5C

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1363
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

RETURN

FUNCTION

Allows the user to return through several levels of subroutine calls.

USAGE (arg1, arg2)

Pos	Argument	Type	Size	Comments
arg1	Level Count	Integer	4	Number of levels to pass through. If zero, the subroutine does a simple return. If positive, the subroutine returns to that number of levels from the calling program. However, it always stops at either the Command Processor or the next lower LINK level, if this argument is too large.
arg2	Ret. Code	Integer	4	Return code from the calling program. Optional. (0 if omitted.)

NOTES

1. This subroutine can be used in a program that has several subroutine layers; when called from an inner routine, it allows the program to return to an outer level, bypassing intermediate levels. This is typically done when an error condition exists and a user wants to bypass further processing and return directly to another step (e.g., an initial menu or error handler).

2. Note that the RETURN subroutine can operate only within subroutine levels of the same program (the same object file). If the level count is larger than the current nesting level of subroutine CALLs, it causes an unlink back to the linking program (or Command Processor). It does not go any further, however, regardless of level count (thus, it can never interfere with the logic of any program other than the user's own).

RETURN Subroutine — A COBOL Example

In the following three programs, control passes from RETURN1 to RETURN2 to RETURN3 via the CALL statement. It then passes from RETURN3 to RETURN1, bypassing RETURN2, via the RETURN subroutine called from RETURN3.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. RETURN1.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600 MAIN-PARAGRAPH.
000700     DISPLAY ''THIS IS LEVEL 1.''.
000800     CALL ''RETURN2''.
000900*THE NEXT STATEMENT WILL BE EXECUTED AFTER THE RETURN SUBROUTINE
001000*PASSES CONTROL BACK TO RETURN1C FROM RETURN3C.
001100     DISPLAY ''THIS IS LEVEL 1 AGAIN.''.
001200     STOP RUN.
```

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. RETURN2.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600 MAIN-PARAGRAPH.
000700     DISPLAY ''THIS IS LEVEL 2.''.
000800     CALL ''RETURN3''.
000900*THE NEXT STATEMENT WOULD BE EXECUTED IF CONTROL WERE PASSED BACK
001000*TO THIS LEVEL FROM RETURN3.
001100     DISPLAY ''THIS IS LEVEL 2 AGAIN.''.
001200 GOBACK.
001300     EXIT PROGRAM.
```

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. RETURN3.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
000700*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
000800*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
000900*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001000*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001100 01  LEVEL-COUNT.
001200     03 FILLER USAGE IS BINARY VALUE 0.
001300*THE NEXT ITEM IS INITIALIZED TO 2 IN ORDER TO INSTRUCT THE
001400*RETURN SUBROUTINE TO PASS CONTROL BACK THAT MANY LEVELS.
001500     03 LEVELCOUNT USAGE IS BINARY VALUE 2.
001600 PROCEDURE DIVISION.
001700 MAIN-PARAGRAPH.
001800     DISPLAY ''THIS IS LEVEL THREE.''.
001900     CALL ''RETURN'' USING LEVEL-COUNT.
002000 GOBACK.
002100     EXIT PROGRAM.
```

SCRATCH

FUNCTION

Provides the ability to scratch a file or library. It has the options of bypassing expiration date checking and limiting access rights for a program with special privileges (as described in system security documentation).

USAGE (arg1, ..., arg7)

Pos	Argument	Type	Size	Comments
arg1	Type	Alpha	1	Type of scratch: F = File scratch L = Library scratch
arg2	File Name	Alpha	8	File to be scratched. Must be included, but ignored if arg1=L.
arg3	Library	Alpha	8	Scratch library: *Arg1=F:* Library where file resides *Arg1=L:* Library to scratch
arg4	Volume	Alpha	6	Volume where library resides.
arg5	Expiration Check	Alpha	1	Indicates whether or not to bypass expiration date checking: B = Bypass checking Blank/omitted = No bypass Optional. Must be included if arg6 is included.
arg6	Access Limit Flag	Alpha	1	Access rights for the file or library: L = Restrict access rights Blank/omitted = Full access The program cannot scratch a file or a library containing a file that the user does not have access rights to. Optional. If present, arg5 must be included.
arg7	Ret. Code	Integer	4	Error return code. See Table 3-14 below.

NOTES

1. Scratching the only file in a library results in scratching the library.

2. For FORTRAN programs, the name of this subroutine must be specified as SCRTCH.

Table 3-14. SCRATCH Error Return Codes

Return Code	Meaning
0	File or library scratched from volume.
4	Volume not mounted.
8	Volume used exclusively by another user.
12	All buffers in use, no scratch.
16	Library not found.
20	File not found.
24	Update access to file protection class denied (single file scratch only).
28	Unexpired file, no scratch (single file scratch only).
32	File in use, no scratch.
36	VTOC error. FDX1 and FDX2 do not agree.
40	VTOC error. FDX2 and FDR do not agree.
44	Invalid argument list address.
48	I/O error. VTOC unreliable.
52	Open, protected, and/or unexpired file(s) bypassed in scratching library.

SCRATCH Subroutine — A COBOL Example

This program allows the user to scratch a file or library, bypassing the check of the expiration period.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. SCRATCHC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FUNCTION-TYPE  PIC X.
000700 77  FILE-NAME  PIC X(8).
000800 77  LIB-RARY  PIC X(8).
000900 77  VOL-UME  PIC X(6).
001000 77  EXPIRE-CHECK PIC X VALUE ''B''.
001100*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001200*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001300*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001400*BYTES FOR THE INTEGER.
001500 01  RETURNCODE.
001600     03 FILLER USAGE IS BINARY VALUE ZERO.
001700     03 ERROR-CODE USAGE IS BINARY.
001800 PROCEDURE DIVISION.
001900 FIRST-PARAGRAPH.
002000     ACCEPT FUNCTION-TYPE, FILE-NAME, LIB-RARY, VOL-UME.
002100     IF FUNCTION-TYPE = ''Z'' GO TO EXIT-PARAGRAPH.
002200     PERFORM CALL-PARAGRAPH.
002300 CALL-PARAGRAPH.
002400     CALL ''SCRATCH'' USING FUNCTION-TYPE, FILE-NAME, LIB-RARY,
002500       VOL-UME, EXPIRE-CHECK, RETURNCODE.
002600     IF ERROR-CODE NOT = 0 DISPLAY ''RETURN CODE = ''ERROR-CODE,
002700       GO TO EXIT-PARAGRAPH.
002800     DISPLAY ''TO VERIFY USE PF KEY 5 FROM THE COMMAND PROCESSOR.''
002900     GO TO FIRST-PARAGRAPH.
003000 EXIT-PARAGRAPH.
003100     STOP RUN.
```

SCRATCH Subroutine — A FORTRAN Example

This program allows the user to scratch a file or library. The user must provide the file, library, and volume names.

```
C  'FNAME', 'LNAME', AND 'VNAME' ARE FILE, LIBRARY, AND VOLUME NAMES
     REAL*8 FNAME, LNAME, VNAME
     LOGICAL*1 OPTION, EXPIRE, ACCESS
     INTEGER RCODE
C  USER PROVIDES NECESSARY FILE, LIBRARY, VOLUME NAMES
     WRITE(0,101) ' ENTER F TO SCRATCH FILE, L TO SCRATCH LIBRARY'
     READ(0,102) OPTION
     IF(OPTION .EQ. 'F') WRITE(0,101) ' ENTER FILE NAME'
     IF(OPTION .EQ. 'F') READ(0,103) FNAME
     WRITE(0,101) ' ENTER LIBRARY NAME'
     READ(0,103) LNAME
     WRITE(0,101) ' ENTER VOLUME NAME'
     READ(0,104) VNAME
C  SET EXPIRATION DATE AND ACCESS LIMITATION OPTIONS
     EXPIRE = 'B'
     ACCESS = ' '
C
C  CALL SCRATCH SUBROUTINE (SCRTCH IN FORTRAN)
     CALL SCRTCH (OPTION, FNAME, LNAME, VNAME, EXPIRE, ACCESS, RCODE)
C
C  WRITE RETURN CODE TO WORKSTATION
     WRITE(0,105) RCODE
 101 FORMAT(A50)
 102 FORMAT(A1)
 103 FORMAT(A8)
 104 FORMAT(A6)
 105 FORMAT(1X,'RETURN CODE = ',I4)
     PAUSE
     END
```

SCRATCH Subroutine — An RPG II Example

This program allows the user to scratch any file or library on the system. The program displays the return code if it is greater than 0.

```
00100FDISPLAY DD  F                     WS
00200C                    ACCPTSCR1
00201C*
00202C*          *** PREPARE PARAMETERS TO BE PASSED ***
00203C*
00210C            FILE      COMP '        '                 88
00220C   88                 MOVE 'L'        TYPE      1
00230C  N88                 MOVE 'F'        TYPE      1
00231C                      Z-ADD0          RCODE    40
00232C*
00233C*           *** EXIT TO RPGCALL MACRO ***
00234C*
00240C                      EXIT RPGSCR
00250C                      RLABL           TYPE
00255C                      RLABL           FILE
00260C                      RLABL           LIBR
00265C                      RLABL           VOLM
00270C                      RLABL           RCODE
00271C*
00272C*             *** TEST RETURN CODE ***
00273C*
00275C            RCODE     COMP 0                      99
00280C   99                 ACCPTSCR2
00282C                      SETON                       LR
00300WSCR1
00400W                 0507       'WHICH FILE DO YOU WISH'
00500W                 0529       ' TO SCRATCH?'
00510W                 0607       '(LEAVE FILE ENTRY BLAN'
00520W                 0629       'K TO SCRATCH AN ENTIRE'
00530W                 0651       ' LIBRARY)'
00600W                 0815       'FILE:'
00700W                 0830                           FILE   8
00800W                 0915       'LIBRARY:'
00900W                 0930                           LIBR   8
01000W                 1015       'VOLUME:'
01100W                 1030                           VOLM   6
01200WSCR2
01300W                 0707       'RETURN CODE IS'
01400W                 0725RCODE
01500W                 0907       'PRESS ENTER TO END JOB'
```

RPGSCR:

```
        RPGCALL   NAME=RPGSCR,CALL=SCRATCH,TYPE,FILE,LIBR,VOLM,        C
              (RCODE,4,F)
```

SEARCH

FUNCTION

Performs a binary search for a particular element in a specified table and indicates whether the element exists in the table.

USAGE (arg1, ..., arg6)

Pos	Argument	Type	Size	Comments
arg1	Table	Alpha	var	Input table to be searched.
arg2	Table Size	Integer	4	Number of items in the input table.
arg3	Item Length	Integer	4	Length of each table item. Range is 1 to 256.
arg4	Search Item	Alpha	var	Value to be searched for in the table.
arg5	Search Item Length	Integer	4	Effective length to be used in searching for the supplied item in the table. Specifying a value less than the item length (arg3) allows the search to match fewer than the entire item length. If omitted, the item length (arg3) is assumed.
arg6	Ret. Code	Integer	4	If the search is successful, this is the index of the item found in the table. If unsuccessful, its value is 0.

NOTES

1. The table should be in either ascending or descending order. SEARCH might not correctly handle tables whose entries are not in ascending or descending order.

2. If the table contains duplicate entries, the binary search might not find the first occurrence of the item in the table.

SEARCH Subroutine — A COBOL Example

This program allows the user to search a five-item table of names to find the location of a specified name.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  SEARCHC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 01  NAMES-LIST.
000700     03 FILLER PIC X(10) VALUE ''ADAMS''.
000800     03 FILLER PIC X(10) VALUE ''BROWN''.
000900     03 FILLER PIC X(10) VALUE ''CUNNINGHAM''.
001000     03 FILLER PIC X(10) VALUE ''DESMOND''.
001100     03 FILLER PIC X(10) VALUE ''EDWARDS''.
001200 01  NAMES-TABLE REDEFINES NAMES-LIST.
001300     03 NAMES PIC X(10) OCCURS 5 TIMES.
001400*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001500*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001600*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001700*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
001800*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
001900 01  TABLE-SIZE.
002000     03 FILLER USAGE IS BINARY VALUE ZERO.
002100     03 TABLE-COUNT USAGE IS BINARY VALUE 5.
002200 01  TABLE-ITEM-LENGTH.
002300     03 FILLER USAGE IS BINARY VALUE ZERO.
002400     03 TABLE-ENTRY-LENGTH USAGE IS BINARY VALUE 10.
002500 77  SEARCH-ITEM PIC X(10).
002600 01  LOCATION.
002700     03 FILLER USAGE IS BINARY VALUE ZERO.
002800     03 INDEX-OF-ITEM USAGE IS BINARY.
002900 PROCEDURE DIVISION.
003000 START-PARAGRAPH.
003100     PERFORM MAIN-PARAGRAPH UNTIL SEARCH-ITEM = ''Z''.
003200     GO TO EXIT-PARAGRAPH.
003300 MAIN-PARAGRAPH.
003400     ACCEPT SEARCH-ITEM.
003500     IF SEARCH-ITEM = ''Z'' GO TO EXIT-PARAGRAPH.
003600     CALL ''SEARCH'' USING NAMES-TABLE, TABLE-SIZE,
003700        TABLE-ITEM-LENGTH, SEARCH-ITEM, LOCATION.
003800     DISPLAY INDEX-OF-ITEM.
003900 EXIT-PARAGRAPH.
004000     STOP RUN.
```

SEARCH Subroutine — A FORTRAN Example

This program allows the user to search a table of color names for an value that the user enters and indicates its position within the table.

```
      REAL*8 TABLE(13), NAME
      INTEGER RCODE
      DATA TABLE/'BLACK','BLUE','BROWN','GOLD','GREEN','GREY',
     1   'ORANGE','PURPLE','RED','SILVER','TAN','WHITE','YELLOW'/
C  ASK USER FOR A COLOR TO FIND
      WRITE(0,101)' ENTER COLOR TO FIND (ENTER STOP TO QUIT)'
      READ(0,102) NAME
C  ENTERING STOP TERMINATES THE PROGRAM
      IF(NAME .EQ. 'STOP') GO TO 99
C  SET TABLE SIZE AND ELEMENT LENGTH
      ISIZE = 13
      LENGTH = 8
C
C  CALL SEARCH SUBROUTINE
      CALL SEARCH (TABLE, ISIZE, LENGTH, NAME, RCODE)
C
C  DISPLAY THE RETURN CODE ON THE WORKSTATION
      WRITE(0,103) RCODE
  101 FORMAT(A41)
  102 FORMAT(A8)
  103 FORMAT(1X, 'RETURN CODE = ',I3)
   99 PAUSE
      END
```

FUNCTION

Sets any of the allowable defaults that are available through the Command Processor SET Usage Constants function.

USAGE (key1, rec1, ..., keyn, recn)

Arguments are specified in keyword-receiver pairs, where the keyword selects the default and the receiver specifies its new value. The user can specify any number of pairs, but each keyword must be immediately followed by a receiver.

Each keyword is a 2-byte alpha value. The keywords, their associated receivers, and the defaults to be set are provided below.

Keyword	Recr Type	Recr Size	Receiver Value
FC	Alpha	1	Default file protection class.
FN	Integer	4	Default printer form number (0-255).
IL	Alpha	8	Default input library.
IV	Alpha	6	Default input volume.
JC	Alpha	1	Default job class for background processing.
JL	Integer	4	Default CPU time limit, in seconds, for background processing.
JS	Alpha	1	Default job status for background processing.
LI	Integer	4	Default lines per page for printer output.
OL	Alpha	8	Default output library.
OV	Alpha	6	Default output volume.
PC	Alpha	1	Default print class (A-Z).
PL	Alpha	8	Default program library (current). See Note.
PM	Alpha	1	Default print mode (S, H, K, or O).
PR	Integer	4	Default printer number for online printing.
PV	Alpha	6	Default program volume (current). See Note.
RL	Alpha	8	Run library (initial). See Note.
RV	Alpha	6	Run volume (initial). See Note.
SV	Alpha	6	Default spool volume.
WV	Alpha	6	Default work volume.

NOTE

"Current" refers to the library or volume that applies to the program containing the call to SET. "Initial" refers to the default library or volume which, when set, applies to the entire session.

SET Subroutine — A COBOL Example

This program allows the user to set the default file protection class, lines-per-page for printer output, and print mode. The user enters the desired values via the ACCEPT statement. Since ACCEPT transfers alphanumeric data only, a BASIC subroutine using the CONVERT statement is called to convert the input for lines-per-page from alphanumeric to integer data. This is explained in Section 2.2.2.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. SETC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 77  FILE-CODE  PIC X(2) VALUE ''FC''.
000700 77  FILE-CLASS  PIC X.
000800 77  LINES-CODE  PIC X(2) VALUE ''LI''.
000900*THE NEXT ITEM RECEIVES THE INPUT FOR LINES-PER-PAGE AND PASSES
001000*IT TO THE BASIC SUBROUTINE.
001100 01  LINES-VALUE.
001200     03 SIGN-ITEM PIC X VALUE ''+''.
001300     03 LINES-NUM PIC X (8).
001400*THE NEXT ITEM RECEIVES THE CONVERTED LINES-PER-PAGE AND PASSES
001500*IT TO THE SET SUBROUTINE.
001600 01  LINES-PER PIC X(4).
001700 77  PRINT-MODE-CODE PIC X(2) VALUE ''PM''.
001800 77  PRINT-MODE PIC X.
001900 PROCEDURE DIVISION.
002000 MAIN-PARAGRAPH.
002100     DISPLAY ''TYPE IN FILE-CLASS, LINES-NUM, PRINT-MODE.''.
002200     ACCEPT FILE-CLASS, LINES-NUM, PRINT-MODE.
002300*THE NEXT STATEMENT CALLS THE BASIC SUBROUTINE.  SEE SECTION
002400*2.2.2 FOR THE BASIC CODE.
002500     CALL ''9TO4'' USING LINES-VALUE, LINES-PER.
002600     CALL ''SET'' USING FILE-CODE, FILE-CLASS, LINES-CODE,
002700         LINES-PER, PRINT-MODE-CODE, PRINT-MODE.
002800     DISPLAY ''TO VERIFY RESULTS, USE PF KEY 2 FROM THE COMMAND PRO
002900-      ''CESSOR.''.
003000     STOP RUN.
```

SET Subroutine — An RPG II Example

This program allows the user to set default input and output libraries and volumes, as well as print class and print mode. The program displays a screen confirming that the parameters have been set as requested.

```
00100FDISPLAY DD  F                   WS

00200C                    ACCPTSCR1
00202C*
00204C*       *** PREPARE PARAMETERS TO PASS TO RPGCALL MACRO ***
00206C*
00210C                    MOVE 'IL'      IL      2
00220C                    MOVE 'IV'      IV      2
00230C                    MOVE 'OL'      OL      2
00240C                    MOVE 'OV'      OV      2
00250C                    MOVE 'PC'      PC      2
00255C                    MOVE 'PM'      PM      2
00256C*
00257C*                *** EXIT TO RPGCALL MACRO ***
00258C*
00260C                    EXIT RPGSET
00265C                    RLABL          IL
00270C                    RLABL          IV
00275C                    RLABL          OL
00280C                    RLABL          OV
00282C                    RLABL          PC
00284C                    RLABL          PM
00286C                    RLABL          LIBIN
00288C                    RLABL          VOLIN
00290C                    RLABL          OUTLB
00291C                    RLABL          OUTVL
00292C                    RLABL          CLASS
00293C                    RLABL          MODE
00294C                    ACCPTSCR2
00295C                    SETON                      LR
00300WSCR1
00400W              0707       'THIS PROGRAM WILL SET '
00500W              0729       'DEFAULTS FOR THE PARAM'
00600W              0751       'ETERS LISTED BELOW.'
00700W              0807       'FILL IN THE VALUES AND'
00800W              0829       ' PRESS ENTER.'
01000W              1215       'INPUT LIBRARY'
01100W              1240                                LIBIN  8
01200W              1315       'INPUT VOLUME'
01300W              1340                                VOLIN  6
01400W              1415       'OUTPUT LIBRARY'
01500W              1440                                OUTLB  8
01600W              1515       'OUTPUT VOLUME'
01700W              1540                                OUTVL  6
01800W              1615       'PRINT CLASS (A TO Z)'
01900W              1640                                CLASS  1
02000W              1715       'PRINT MODE (S,H,O)'
02100W              1740                                MODE   1
```

```
02200WSCR2
02300W                  0707        'PARAMETERS SET AS REQU'
02400W                  0729        'ESTED.  PRESS ENTER TO'
02500W                  0751        ' END JOB.'
```

RPGSET:

```
          RPGCALL   NAME=RPGSET,CALL=SET,IL,LIBIN,IV,VOLIN,OL,OUTLB,     C
                OV,OUTVL,PC,CLASS,PM,MODE
```

SORT

FUNCTION

Sorts a character array on a specified field, in either ascending or descending order. Output from the subroutine can be either the sorted array or a locator-type array. A locator-type array contains pointers to the elements in the array and indicates the sorted order of those elements.

USAGE (arg1, ..., arg9)

Pos	Argument	Type	Size	Comments
arg1	Input	Alpha	var	Input array to be sorted.
arg2	Elements	Integer	4	Number of elements in the input array. Range is 0 to 32767.
arg3	Element Length	Integer	4	Length of each element in the array. Range is 1 to 256.
arg4	Output	Alpha	var	Output array to receive the sorted values or pointers (if locator type sort - see arg8). If omitted, the sorted elements are placed in the input array (arg1).
arg5	Start	Integer	4	Starting position of the sort field in the element. Default is character 1.
arg6	Sort Length	Integer	4	Length of the sort field. Standard sort — a 255-byte sort field cannot be used with a 256-byte record. Locator-type sort — the sort length plus the locator size cannot exceed 256 bytes. Default is to sort the entire record.
arg7	Sort type	Alpha	1	Type of sort to be performed: A = Ascending (default) D = Descending
arg8	Locator Flag	Alpha	1	Flag for locator (addrout) type sort: S = Standard sort (default) L = Locator type sort
arg9	Locator Length	Integer	4	Desired size of each locator element. Range is 1 to 4. Default is 2.

NOTES

1. Arguments 4 through 9 are optional; however, if one is present, all previous arguments must be included.

2. No check is made for appropriate locator element size (e.g., locator length of 1 would be insufficient for an input array with more than 255 elements).

SORT Subroutine — A FORTRAN Example

This program allows the user to perform a standard sort, in ascending order, on a table of 4-character values read from an external data file. Arguments 5 through 9 are omitted because the sort starts in column 1 and affects the entire record.

```
C  'ARRAY1' CONTAINS THE UNSORTED TABLE
C  'ARRAY2' CONTAINS THE SORTED TABLE
      DIMENSION ARRAY1(12), ARRAY2(12)
C  READ TABLE OF 12 VALUES FROM DATA FILE
      DO 1 I=1,12
    1 READ(2,101) ARRAY1(I)
C
C  CALL SORT SUBROUTINE
      CALL SORT(ARRAY1, 12, 4, ARRAY2)
C
C  DISPLAY BOTH TABLES ON THE WORKSTATION
      WRITE(0,102) ARRAY1, ARRAY2
  101 FORMAT(A4)
  102 FORMAT(' UNSORTED:'/3(1X,4(1X,A4)/)/' SORTED:'/3(1X,4(1X,A4)/))
      PAUSE
      END
```

STRING

FUNCTION

Performs the following string manipulation functions:

1. Moves a string to another variable and pads it with a user-specified character.
2. Moves a portion of a string to another variable.
3. Centers a string.
4. Left- or right-justifies a string.
5. Reverses the order of characters in a string.
6. Translates the string according to a selected or user-specified translation table.

USAGE (arg1, arguments)

Arg1 defines the string function and determines the number and nature of the additional arguments.

1. Move a string and pad it with a user-specified character

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is MV
arg2	Input	Alpha	var	String to be processed.
arg3	Input Length	Integer	4	Length of input string.
arg4	Output	Alpha	var	Output location for the moved string.
arg5	Output Length	Integer	4	Length of output string. If omitted, assumed to be the input string length. Must be present if arg6 is included.
arg6	Pad Character	Alpha	1	Character to be used as the pad if the output length (arg5) is greater than the input length (arg3). If omitted, blank (hex 20) is assumed. If included, arg5 must also be present.

2. Move a contiguous string of characters from one location to another (move-indexed)

Same as MV, but includes an offset for input and output locations (primarily for BASIC STR emulation).

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is MI
arg2	Input	Alpha	var	Input string to process.
arg3	Input	Integer Index	4	Offset (from 0) of the first character of the input string to be moved.
arg4	Input Length	Integer	4	Number of characters to be moved, starting with the character indicated by arg3.
arg5	Output	Alpha	var	Output location for the moved string.
arg6	Output Index	Integer	4	Offset within the output string to move string to. Optional. If omitted, offset 0 is assumed.
arg7	Output Length	Integer	4	Length of the output string.If omitted, length of the input string assumed. If present, the program must include arg6.
arg8	Pad	Alpha Character	1	Character to be used as the pad if the output length (arg7) exceeds the input length (arg4). If omitted, blank (hex 20) is assumed. If present, the program must include arg6 and arg7.

3. Center, left- or right-justify, or reverse the characters in the input string

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	String manipulation function: CT = Center LJ = Left-justify RJ = Right-justify RV = Reverse
arg2	Input	Alpha	var	Input string to process.
arg3	Length	Integer	4	Length of the input string.
arg4	Output	Alpha	var	Output location for the shifted characters. Length is the same as that of the input string. Optional. If omitted, assumed to be the same as the input string (the string function is performed "in place").

4. Translate the input string with a user-supplied translation table

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is TT
arg2	Input	Alpha	var	Input string to process.
arg3	Length	Integer	4	Length of the input string.
arg4	Translate Table	Alpha	256	Table to be used for the translation. The character in the input string whose binary value is N is translated into the character in position (N+1) in the table. (See an ASCII Collating Sequence table for binary values of ASCII characters.)
arg5	Output	Alpha	var	Input string translation. Optional. If omitted, the input string contains the translation.

5. Translate the input string with a user-supplied translation list

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Value is TL
arg2	Input	Alpha	var	Input string to process.
arg3	Length	Integer	4	Length of the input string.
arg4	Translate List	Alpha	var	List of to/from character pairs used in the translation. Indicate the end of the list by the pair X'0000'. In each byte pair in the translation list, all occurrences in the input string of the character indicated by the second byte are translated to the character indicated by the first byte. Any input characters not represented in the list are not changed in the translation.
arg5	Output	Alpha	var	Input string translation. Optional. If omitted, the input string contains the translation.

6. Translate ASCII input to EBCDIC or translate EBCDIC input to ASCII

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Translation function: AE = ASCII to EBCDIC EA = EBCDIC to ASCII
arg2	Input	Alpha	var	Input string to translate.
arg3	Length	Integer	4	Length of the input string.
arg4	Output	Alpha	var	Output location for translated characters. Length is the same as that of the input string. Optional. If omitted, the length is assumed to be the same as the input string (translation is performed "in place").

NOTES

1. If the input and output locations are the same, the functions are performed "in place."

2. With the exception of the MV and MI functions, the results are not guaranteed to be correct if the input and output locations are different but overlap in some other way.

3. The MV and MI functions are always performed one byte at a time, from left to right. Thus, overlapping operands result in either "correct" moves or character propagation, depending on the type of overlap. This is similar to the way in which the BASIC "COPY" instruction operates.

STRING Subroutine — A FORTRAN Example

This example demonstrates the center string (CT) and reverse string (RV) functions. Results are shown after the program.

```
      REAL*8 CHARS1, CHARS2
      CHARS1 = 'ABCD    '
      LENGTH = 8
C
C  CALL STRING TO CENTER CHARS1 - RESULT IS CHARS2
      CALL STRING ('CT', CHARS1, LENGTH, CHARS2)
C
      WRITE(0,101) '     12345678  12345678'
      WRITE(0,102) ' CT', CHARS1, CHARS2
C
C  CALL STRING TO REVERSE CHARS1 - RESULT IS CHARS2
      CALL STRING ('RV', CHARS1, LENGTH, CHARS2)
C
      WRITE(0,102) ' RV', CHARS1, CHARS2
  101 FORMAT(A23)
  102 FORMAT(A3, 2(2X, A8))
      PAUSE
      END
```

The output from this program appears like this:

```
    12345678  12345678
CT  ABCD        ABCD
RV  ABCD          DCBA
PAUSE:     0
```

STRING Subroutine — AN RPG II Example

This program asks the user to input a 40-character string and choose a function (center, reverse, left justify, convert ASCII to EBCDIC, or move to a longer string and pad with a chosen character). The program performs the requested function and displays the results. The user can then make another choice.

```
00100FDISPLAY DD  F WS

00110C           TOP       TAG
00120C                     SETOF                     88
00200C                     ENBLEK1,K2,K3
00300C                     ENBLEK4,K5,KG
00310C*
00320C*    *** DISPLAY SCREEN ALLOWING USER TO CHOOSE STRING FUNCTION ***
00330C*                  *** OR TO END THE JOB ***
00340C*
00400C                     ACCPTSCR1
00403C*
00404C*              *** END JOB IF PF 16 WAS PRESSED ***
00406C*
00410C   KG                GOTO END
00420C*
00430C*     *** PREPARE PARAMETERS TO PASS TO RPGCALL MACRO ***
00435C*       *** (FOR ALL FUNCTIONS EXCEPT MOVE) ***
00440C*
00500C                     Z-ADD40        LEN     40
00600C   K5                GOTO MOVE
00700C   K1                MOVE 'CT'      FN       2
00800C   K2                MOVE 'RV'      FN       2
00900C   K3                MOVE 'LJ'      FN       2
01000C   K4                MOVE 'AE'      FN       2
01002C*
01004C*       *** EXIT TO RPGCALL MACRO (FOR ALL FUNCTIONS EXCEPT MOVE)
01006C*
01010C                     EXIT RPGST1
01020C                     RLABL           FN
01030C                     RLABL           STR
01040C                     RLABL           LEN
01041C                     SETON                     88
01050C                     GOTO ANSR
```

```
01060C*
01070C*              *** PERFORM MOVE FUNCTION ***
01080C*
01100C            MOVE      TAG
01200C                      MOVE 'MV'       FN      2
01210C                      MOVE ' '        OSTR    70
01220C                      Z-ADD70         OLEN    40
01300C                      EXIT RPGST2
01610C                      RLABL           OSTR
01620C                      RLABL           OLEN
01630C                      RLABL           PAD
01640C*
01650C*           *** DISPLAY RESULT OF STRING MANIPULATION ***
01660C*
02110C            ANSR      TAG
02120C                      ACCPTSCR2
02130C                      GOTO TOP
02140C            END       TAG
02150C                      SETON                       LR

02200WSCR1
02300W                  0707        'PLEASE ENTER A CHARACT'
02400W                  0729        'ER STRING AND CHOOSE A'
02500W                  0751        ' FUNCTION.'
02600W                  1015                                STR   40
02700W                  1210        'PF 1 - CENTER'
02800W                  1310        'PF 2 - REVERSE'
02900W                  1410        'PF 3 - LEFT JUSTIFY'
03000W                  1510        'PF 4 - DISPLAY EBCDIC '
03010W                  1532        'EQUIVALENT'
03100W                  1610        'PF 5 - MOVE TO A LARGE'
03200W                  1632        'R STRING AND PAD WITH '
03300W                  1654        'A SPECIFIED CHARACTER'
03340W                  1715        '(PADDING CHARACTER =  '
03350W                  1736                                PAD     1
03351W                  1737        ')'
03355W                  2007        'PRESS PF 16 TO END JOB'
03400WSCR2
03410W        88        0702STR                             STR
03500W       N88        0702OSTR                            OSTR
03600W                  1007        'PRESS ENTER TO TRY AGA'
03700W                  1029        'IN.'
```

RPGST1:

```
        RPGCALL   NAME=RPGST1,CALL=STRING,FN,STR,(LEN,4,F)
```

RPGST2:

```
        RPGCALL   NAME=RPGST2,CALL=STRING,FN,STR,(LEN,4,F),OSTR,      C
              (OLEN,4,F),PAD
```

SUBMIT

FUNCTION

Submits a background job to be run or held for later processing.

USAGE (arg1, ..., arg11)

Pos	Argument	Type	Size	Comments
arg1	File	Alpha	8	Name of the procedure file to be submitted.
arg2	Library	Alpha	8	Library containing the procedure. The default is the PROGLIB value, as defined by PF2 (SET) of the Command Processor.
arg3	Volume	Alpha	6	Volume containing the procedure. The default is the PROGVOL value, as defined by PF2 (SET) of the Command Processor.
arg4	Job Name	Alpha	8	User-supplied name for the job using the submitted procedure. The default is blank.
arg5	Status	Alpha	1	Status of the submitted job: R = Run immediately. H = Hold. Blank = Use the value specified by a SET SVC, a SET Procedure language statement, or by PF2 (SET) of the Command Processor. The default is blank.
arg6	Job Disposition	Alpha	1	Disposition of the job after completion: D = Delete from queue (default). R = Return to queue.
arg7	Job Class	Alpha	1	Job class of the procedure submitted. Must be a letter from A to Z or blank. If blank, use the value specified by the SET SVC, a SET Procedure language statement, or by PF2 (SET) of the Command Processor. The default is blank.
arg8	Abort Action	Alpha	1	Action to take if the job aborts: D = Produce program dump. N = No program dump. R = Produce dump only if requested elsewhere in the program. (Default).
arg9	CPU Time Limit	Integer	4	CPU time limit, in 1/100 seconds: 0 = No time limit (default). -1 = Use the value specified by a SET SVC, a SET Procedure language statement, or PF2 (SET) of the Command Processor.

Pos	Argument	Type	Size	Comments
arg10	Limit Flag	Alpha	1	Action to take if the CPU time limit (arg9) is exceeded: C = Cancel program. P = Pause. W = Continue the procedure, but generate an operator warning. (Default).
arg11	Ret. Code	Integer	4	Error return code. See Table 3-15 below.

Arguments 2 through 10 are optional. If the program uses an argument, all the preceding arguments must be used.

Table 3-15. SUBMIT Error Return Codes

Return Code	Meaning
0	Successful.
8	Volume not mounted.
12	Volume used exclusively by another user.
16	All buffers in use, unable to perform verification.
20	File not found.
24	Improper file type, or the file contains zero records.
28	File access denied.
32	VTOC error. FDX1 and FDX2 do not agree.
36	VTOC error. FDX2 and the FDR1 and FDR2 do not agree.
40	Invalid specification of file, library, and volume.
48	System task not running, no spooled printing or interactive jobs.
52	Error in performing XMIT to system task.
56	Invalid options specified in argument list.

SUBMIT Subroutine — A BASIC Example

This program allows the user to submit any procedure as a background job by specifying the Procedure language file, library, volume, and job names. The program provides default values for status, disposition, abort action, and limit action in lines 1000-1300.

```
000100DIM FILE$          08
000200DIM LIBRARY$       08
000300DIM VOLUME$        06
000400DIM JOBNAME$       08
000500DIM STATUS$        01
000600DIM DISPOSITION$   01
000700DIM JOBCLASS$      01
000800DIM ABORTACTION$   01
000900DIM LIMITACTION$   01
001000STATUS$        =''R''
001100DISPOSITION$   =''D''
001200ABORTACTION$   =''R''
001300LIMITACTION$   =''W''
001400
001500LOOP:
001600GOSUB PUTSCREEN
001700GOSUB DOSUBMIT
001800GOTO LOOP
001900
002000PUTSCREEN:
002100ACCEPT                                                                  !
002200       AT (01,10),                                                      !
002300''Demonstration of Submitting a Background Job (SUBMIT) Subroutine      !
002400'',                                                                     !
002500       AT (05,03),                                                      !
002600''Fill in the information requested below, press ENTER, to submit       !
002700a job.'',                                                               !
002800       AT (07,03),                                                      !
002900''FILE NAME:'',                                                         !
003000       AT (07,17), FILE$          , CH(08),                             !
003100       AT (07,29),                                                      !
003200''(Procedure file to be submitted)'',                                   !
003300       AT (08,03),                                                      !
003400''LIBRARY:'',                                                           !
003500       AT (08,17), LIBRARY$       , CH(08),                             !
003600       AT (09,03),                                                      !
003700''VOLUME:'',                                                            !
003800       AT (09,17), VOLUME$        , CH(06),                             !
003900       AT (10,03),                                                      !
004000''JOB NAME:'',                                                          !
004100       AT (10,17), JOBNAME$       , CH(08),                             !
004200       AT (10,29),                                                      !
004300''(Name of associated background job)'',                                !
```

```
004400       AT (11,03),                                                    !
004500''STATUS:'',                                                          !
004600       AT (11,17), STATUS$       , CH(01),                           !
004700       AT (11,29),                                                    !
004800''(R-run;H-hold)'',                                                   !
004900       AT (12,03),                                                    !
005000''DISPOSITION:'',                                                     !
005100       AT (12,17), DISPOSITION$  , CH(01),                           !
005200       AT (12,29),                                                    !
005300''(D-dequeue;R-requeue)'',                                            !
005400       AT (13,03),                                                    !
005500''JOB CLASS:'',                                                       !
005600       AT (13,17), JOBCLASS$     , CH(01),                           !
005700       AT (14,03),                                                    !
005800''ABORT ACTION:'',                                                    !
005900       AT (14,17), ABORTACTION$  , CH(01),                           !
006000       AT (14,29),                                                    !
006100''(D-program dump;N-no program dump;R-dump on request)'',             !
006200       AT (15,03),                                                    !
006300''CPU LIMIT:'',                                                       !
006400       AT (15,17), CPULIMIT%     , PIC(####),                        !
006500       AT (15,29),                                                    !
006600''(Time limit for CPU usage)'',                                       !
006700       AT (16,03),                                                    !
006800''LIMIT ACTION:'',                                                    !
006900       AT (16,17), LIMITACTION$  , CH(01),                           !
007000       AT (16,29),                                                    !
007100''(C-cancel program;P-pause;W-warning message)'',                     !
007200       AT (18,03),                                                    !
007300''RETURN CODE:'',                                                     !
007400       AT (18,17), RETURNCODE%   , PIC(##)
007500RETURN
007600
007700DOSUBMIT:
007800   CALL ''SUBMIT'' ADDR(FILE$,LIBRARY$,VOLUME$,                       !
007900                        JOBNAME$,STATUS$,DISPOSITION$,                !
008000                        JOBCLASS$,ABORTACTION$,CPULIMIT%,             !
008100                        LIMITACTION$,RETURNCODE%)
008200RETURN
```

FUNCTION

Allows the user to reserve or release a device or peripheral processor on the system.

USAGE (arg1, ..., arg3)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	2	Function code: D+ = Reserve the device D− = Release the device P+ = Reserve the peripheral processor P− = Release the peripheral processor
arg2	Unit No.	Integer	4	Number of the device or peripheral processor. It must be nonnegative (only values 0-255 are recognized; larger values produce an error return code).
arg3	Ret. Code	Integer	4	Error return code. See Table 3-16 below.

Table 3-16. UNITRES Error Return Codes

Return Code	Meaning
0	Successful reserve/release.
4	Invalid unit address in argument list.
8	Invalid function code in argument list.
12	Invalid unit type in argument list.
16	(Reserved)
20	PP specified for nonprogrammable device.
24	PP reservation conflict.
28	(Reserved)
32	Release specified for a device or PP that the caller does not own.
36	Invalid device type.
40	Device reservation conflict.

UNITRES Subroutine — A COBOL Example

This program allows the user to reserve and then release a device or peripheral processor interactively by entering the unit number and type (D or P) at the workstation. Since the COBOL ACCEPT statement transfers only alphanumeric data, this program calls the BASIC subroutine 9TO4, discussed in Section 2.2.2, to convert the entered unit number to a format that the UNITRES subroutine can use.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. UNITRESC.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600 01  FUNCTION.
000700     03  FUNCTION-NAME PIC X.
000800     03  FUNCTION-SIGN PIC X VALUE ''+''.
000900*THE NEXT ITEM PASSES THE UNIT NUMBER TO THE BASIC SUBROUTINE.
001000 01  UNIT-NUMBER.
001100     03 SIGN-ITEM PIC X VALUE ''+''.
001200     03 UNIT-VALUE PIC X(8).
001300*THE NEXT ITEM RECEIVES THE CONVERTED UNIT NUMBER FROM THE BASIC
001400*SUBROUTINE
001500 01  UNIT-INTEGER PIC X(4).
001600*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001700*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
001800*HALFWORD-BINARY ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
001900*BYTES FOR THE INTEGER.
002000 01  RETURN-KODE.
002100     03 FILLER USAGE IS BINARY VALUE ZERO.
002200     03 ERROR-CODE USAGE IS BINARY.
002300 PROCEDURE DIVISION.
002400 MAIN-PARAGRAPH.
002500     ACCEPT UNIT-VALUE.
002600     CALL ''9TO4'' USING UNIT-NUMBER, UNIT-INTEGER.
002700     ACCEPT FUNCTION-NAME.
002800     DISPLAY ''PRESS ENTER TO RESERVE UNIT ''UNIT-VALUE.
002900     PERFORM CALL-PARAGRAPH.
003000     MOVE ''-'' TO FUNCTION-SIGN.
003100     DISPLAY ''PRESS ENTER TO RELEASE UNIT ''UNIT-VALUE.
003200     PERFORM CALL-PARAGRAPH.
003300     GO TO EXIT-PARAGRAPH.
003400 CALL-PARAGRAPH.
003500     CALL ''UNITRES'' USING FUNCTION, UNIT-INTEGER, RETURN-KODE.
003600     IF ERROR-CODE NOT = 0 DISPLAY ''ERROR-CODE = ''ERROR-CODE,
003700         GO TO EXIT-PARAGRAPH.
003800     DISPLAY ''TO VERIFY RESULT USE PF KEY 6 FROM THE COMMAND
003900-        '' PROCESSOR.''.
004000 EXIT-PARAGRAPH.
004100     STOP RUN.
```

UPDATFDR

FUNCTION

Allows the user to change attributes of a file or library. The attributes are listed below.

USAGE (arg1, ..., arg5, arg6 [repeatable keyword-value pairs], ..., arg8)

Pos	Argument	Type	Size	Comments
arg1	Update Range	Alpha	1	Specifies range of the update: F = Update single file L = Update all files in a library
arg2	File Name	Alpha	8	File to be modified. Ignored if arg1 =L.
arg3	Library	Alpha	8	Library.
arg4	Volume	Alpha	6	Volume.

The program can use the following two arguments as optionally repeatable keyword-value pairs.

Pos	Argument	Type	Size	Comments
arg5	Keyword	Alpha	2	File attribute to be changed.
arg6	Value	Alpha	var	New value.

Keyword	Recr Type	Recr Size	Receiver Value
CD	Alpha	6	Creation date in the form YYMMDD.
ED	Alpha	6	Expiration date in the form YYMMDD.
FC	Alpha	1	File protection class.
ID	Alpha	3	Owner's ID.
MD	Alpha	6	Last modification date in the form YYMMDD
ME	Alpha	4	Special execute access flags. See Note 3.
MR	Alpha	4	Special read access flags. See Note 3.
MW	Alpha	4	Special write access flags. See Note 3.
RS			Value ignored. Release unused space in the file(s).

Pos	Argument	Type	Size	Comments
arg7	Access Limit Flag	Alpha	1	Specifies access rights: L = Restricted to the user's access rights Blank or omitted = No restriction (use the special access rights of the program, if available) Optional.

Pos	Argument	Type	Size	Comments
arg8	Ret. Code	Integer	4	Error return code. Nonzero value depends on the value of arg1: *Arg1 = F:* Return codes as follows: 4-96= UPDATFDR return codes (see Table 3-17 below) 104-196= Ret. Code for READFDR + 100 *Arg1 = L:* Additional return codes: 100= One or more files could not be updated (for any reason) 204-296= Ret. Code for READVTOC + 200

NOTES

1. Return codes are structured as described in the arg8 description for these reasons: for single-file updates, READFDR is called; for library updates, READFDR and READVTOC are both called.

2. A "blocks-lost" condition, indicated by return code 44, is not detected if arg1=L.

3. The ME, MR, and MW keywords require that the user have security administrator rights. The remaining keywords require only that the user be the creator of the file or files to be modified.

4. For FORTRAN programs, the name of this subroutine must be specified as UPDFDR.

Table 3-17. UPDATFDR Error Return Codes

Return Code	Meaning
0	File label updated.
4	All buffers in use, no update.
8	Volume not mounted, no update.
12	Volume used exclusively by another user.
16	Wrong disk type, no update.
20	File not open in an exclusive mode for group 1, group 2, and/or group 3, no update.
24	Library not found.
28	File not found.
32	Update access to this file protection class denied, no update.
36	File not closed for group 4 and/or group 5, no update.
40	VTOC full, no spare for FDR2 label.
44	VTOC full, no spare for freed extent. Extent lost.
48	VTOC error, FDX1 and FDX2 do not agree.
52	VTOC error, FDX2 and FDR do not agree.
56	VTOC error, FDX1 and FDR do not agree.
60	VTOC error, invalid data in FDR1 or FDR2.
64	System/VTOC error, FLUB and FDR1 do not agree.
68	Disk I/O error, VTOC unreliable.
72	Group 5 update attempted on nonprogram file.

UPDATFDR Subroutine — A COBOL Example

This program allows the user to modify the expiration date, file protection class, and owner's ID for individual files or all the files in a library.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. UPDTFDRC
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 WORKING-STORAGE SECTION.
000600*THE FOLLOWING ITEMS ARE THE ARGUMENTS FOR THE UPDATFDR SUBROUTINE
000700 77  UPDATE-RANGE  PIC X(1).
000800 77  FILE-NAME  PIC X(8).
000900 77  LIB-RARY  PIC X(8).
001000 77  VOL-UME  PIC X(6).
001100 77  EXPIRE-KEY  PIC X(2) VALUE ''ED''.
001200 77  EXPIRE-DATE  PIC X(6).
001300 77  PROTECT-KEY  PIC X(2) VALUE ''FC''.
001400 77  FILE-CLASS  PIC X.
001500 77  ID-KEY  PIC X(2) VALUE ''ID''.
001600 77  ID  PIC X(3).
001700 77  LIMIT-FLAG PIC X VALUE '' ''.
001800*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
001900*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
002000*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
002100*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBROUTINE,
002200*INITIALIZE THE HIGH-ORDER BYTES TO ZERO.
002300 01  RETURNCODE.
002400     03 FILLER USAGE BINARY VALUE ZERO.
002500     03 ERROR-CODE USAGE BINARY.
002600 PROCEDURE DIVISION.
002700 MAIN-PARAGRAPH.
002800     ACCEPT UPDATE-RANGE, FILE-NAME, LIB-RARY, VOL-UME,
002900         EXPIRE-DATE, FILE-CLASS, ID.
003000     CALL ''UPDATFDR'' USING UPDATE-RANGE, FILE-NAME, LIB-RARY,
003100         VOL-UME, EXPIRE-KEY, EXPIRE-DATE, PROTECT-KEY,
003200         FILE-CLASS, ID-KEY, ID, RETURNCODE.
003300     DISPLAY ''TO VERIFY RESULTS USE PF KEY 5 FROM THE COMMAND PROC
003400-        ''ESSOR.''.
003500     STOP RUN.
```

FUNCTION

Performs I/O operations at the workstation and returns values associated with those operations.

This subroutine provides a variety of I/O operations. The following options are available in most, but not all, higher-level programming languages:

Open or Close the Workstation file
READ Altered
READ Diagnostic
READ Tabs
WRITE Selected
WRITE Tabs

The *VS Principles of Operation* provides a description of these operations.

USAGE (arg1, arg2, arguments)

Arg1 defines the type of function to be performed, arg2 specifies a file or a User File Block (UFB). Arg1 determines the number and nature of the remaining arguments.

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Type of function to be performed: O = Open the workstation file C = Close the workstation file X = Perform an I/O operation W = Wait for interrupt A = Move AID character
arg2	User File Receiver	Alpha	140	File name (COBOL), file number (BASIC), parameter reference name for a UFB (Assembler), or data item used to hold a UFB address. A data item used to hold a UFB address must have a length of 140 and be fullword aligned before the file is opened. It can be examined or used at any time between OPEN and CLOSE, but it should not be changed during this time.

The remaining arguments depend on the function type. If the function type is C, no further arguments are necessary.

1. OPEN the Workstation file

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is O
arg2	Device	Integer	4	Device number of workstation to be opened (must be nonnegative). If the device number is 255, the user's workstation is assumed.
arg3	File Recr	Alpha	140	Area to be used as the UFB for the workstation file. It is initialized to valid UFB information prior to OPEN. It can be an FD (COBOL), a file number (BASIC), a UFB block (Assembler), or a variable or array that this subroutine uses to hold the UFB. (If it is the latter, it must be fullword aligned.) It can be examined or used (standard DMS) at any time between OPEN and CLOSE, but should not be erased or otherwise radically changed during this time.
arg4	Ret. Code	Integer	4	Error return code for OPEN operation: 0 = Successful. 4 = Not a workstation. 8 = OPEN error. The OPEN error status can be found in the UFB file status bytes FS1/FS2; either an Open Exit or a Cancel/Respecify exit was taken.

NOTE

Older versions of WSXIO did not require arguments 2 and 4. That argument list will continue to be supported for a limited amount of time; programs using WSXIO with the previous argument list should be updated.

2. Perform an I/O operation (the operations are listed in the description of argument 3)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is X
arg2	File Recr	Alpha	140	As in OPEN.

Pos	Argument	Type	Size	Comments
arg3	Command Code	Alpha	1	Indicates the I/O operation to be performed. Arg3 is a hexadecimal character in the first byte of the I/O Command Word (IOCW). Any arg3 value is accepted; the following should be used to perfoem standard DMS functions: X'40' = READ X'44' = READ tabs X'48' = READ diagnostic X'50' = READ altered X'80' = WRITE X'84' = WRITE tabs X'90' = WRITE selected
arg4	Order Area	Alpha	var	Order area to be transmitted to the workstation for the I/O operation. Provided by the user program.
arg5	Order Area Length	Integer	4	Length of the order area. Value can be 0 to Area 4096. The sum of arg5 and arg7 cannot exceed 4096. Optional. Default is 4 bytes.
arg6	Mapping Area	Alpha	var	Mapping area transmitted to the workstation for the I/O operation, provided by the user program.
arg7	Mapping Area Length	Integer	4	Length of the mapping area. Value can be 0 to Area 4096.
arg8	IOSW Recr	Alpha	8	Data item that receives the I/O Status Word (IOSW) after the I/O operation.

NOTES

1. For READ and WRITE operations, arg4, arg6, and arg7 are mandatory.

2. If possible, the order and mapping areas are sent to the workstation directly from the locations specified by arg4 and arg6; however, in the following situations, the data must be moved to a temporary location for the I/O operation.
   a) When the order and mapping areas do not occupy adjacent locations, and neither length is zero.
   b) When the combined area is not fullword aligned.
   c) When the combined area spans more than 2 contiguous pages of memory. The minimum amount of stack space required to properly align the data is used.

3. Wait for interrupt

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is W Indicates an instruction to wait for an unsolicited interrupt from the workstation.
arg2	File recr	Alpha	140	As in OPEN.
arg3	Timeout Value	Integer	4	Number of 1/100 seconds to wait for an interrupt.
arg4	IOSW recr	Alpha	8	Data item that receives the IOSW after the timeout is taken. If no interrupt occurs before the timeout is taken, the IOSW is unchanged.

4. Return AID character (See Table 3-18 below for a list of AID characters and their meanings.)

Pos	Argument	Type	Size	Comments
arg1	Function	Alpha	1	Value is A Indicates that the AID character is to be moved to the data item referenced by arg3.
arg2	File recr	Alpha	140	As in OPEN.
arg3	AID recr	Alpha	1	Data item that receives the current AID character. This character is also available in the third byte of the IOSW immediately after the I/O operation.

Table 3-18. AID Characters and Their Meanings

AID Character	Hexadecimal Character	ASCII Character
Keyboard unlocked by write	20	(blank)
Keyboard locked by write	21	'
ENTER key	40	@
PF1	41	A
PF2	42	B
PF3	43	C
PF4	44	D
PF5	45	E
PF6	46	F
PF7	47	G
PF8	48	H
PF9	49	I
PF10	4A	J
PF11	4B	K
PF12	4C	L
PF13	4D	M
PF14	4E	N
PF15	4F	O
PF16	50	P
PF17	61	a
PF18	62	b
PF19	63	c
PF20	64	d
PF21	65	e
PF22	66	f
PF23	67	g
PF24	68	h
PF25	69	i
PF26	6A	j
PF27	6B	k
PF28	6C	l
PF29	6D	m
PF30	6E	n
PF31	6F	o
PF32	70	p

WSXIO Subroutine — A COBOL Example

This program opens the workstation file, performs a WRITE to the workstation, allows the user to modify fields already written, and performs a READ ALTERED (which reads into memory only the fields that have been altered). It also erases and protects the screen, performs a WRITE SELECTED (which writes to the screen only the fields that have been altered), and closes the workstation file. The program also displays the workstation's I/O Status Word (IOSW) after calling the subroutine HEXUNPK to convert the IOSW from ASCII characters to hexadecimal digits.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. WSXIOC.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500*THE FOLLOWING ITEMS WILL BE USED FOR ARGUMENT 3 (THE COMMAND
000600*CODE) FOR THE ORDER AREA OF THE SCREEN, AND FOR FIELD ATTRIBUTE
000700*CHARACTERS.
000800 FIGURATIVE-CONSTANTS. WRITE-COMMAND IS ''80'',
000900         FIRST-ORDER   IS ''01A0''
001000         SECOND-ORDER  IS ''0000''
001100         SELECT-COMMAND IS ''90'',
001200         ALTERED-COMMAND IS ''50'',
001300         ERASE-PROTECT IS ''0102''.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600     SELECT SCREEN, ASSIGN TO ''SCREEN'', ''DISPLAY'',
001700     ACCESS MODE IS RANDOM,
001800     PFKEY IS PFKEY-RECEIVE.
001900 DATA DIVISION.
002000 FILE SECTION.
002100 FD SCREEN,
002200     LABEL RECORDS ARE STANDARD.
002300 01  SCREEN-REC.
002400     03 ORDERAREA.
002500        05 ORDER-1  PICTURE IS XX.
002600        05 ORDER-2  PICTURE IS XX.
002700 03 SCREEN-AREA PIC X(1920).
002800 WORKING-STORAGE SECTION.
002900 01  MAPPING-AREA.
003000     03 FILLER PIC X(720) VALUE SPACE.
003100*IN FAC-1 AND FAC-2, FIGURATIVE-CONSTANT ''WRITE-COMMAND'' IS USED
003200*FOR THE BRIGHT-MODIFY FIELD ATTRIBUTE CHARACTER.
003200     03 FAC-1 PIC X VALUE WRITE-COMMAND.
003300     03 FIELD-1 PIC X(49) VALUE ''MODIFY THIS FIELD
003400     03 FILLER PIC X(400) VALUE SPACE.
003500     03 FAC-2 PIC X VALUE WRITE-COMMAND.
003600     03 FIELD-2 PIC X(15) VALUE ''DO NOT MODIFY''.
003700*THE NEXT ITEM IS THE FUNCTION FLAG.  IT IS INITIALIZED TO ''0''
003800*FOR THE FIRST FUNCTION, OPEN.
003900 77  FUNC-FLAG PIC X VALUE ''0''.
```

```
004000*THE NEXT ITEM IS THE COMMAND CODE INITIALIZED TO THE WRITE
004100*COMMAND FOR THE FIRST USE OF THIS ARGUMENT.
004200 77  COMMAND PIC X VALUE WRITE-COMMAND.
004300*AS EXPLAINED IN SECTION 2.2.2, COBOL ACCEPTS HALFWORD INTEGERS
004400*ONLY.  DEFINE A FOUR-BYTE GROUP ITEM TO BE COMPOSED OF TWO
004500*HALFWORD-BINARY, ELEMENTARY ITEMS, AND USE THE LOW-ORDER TWO
004600*BYTES FOR THE INTEGER.  TO PASS AN INTEGER TO THE SUBRDUTINE,
004700*INITIALIZE THE HIGH-ORDER BYTES TD ZERO.
004800 01  ORDER-AREA-LENGTH.
004900     03  FILLER USAGE IS BINARY VALUE IS ZERO.
005000     03  ORDAREA-LENGTH  BINARY VALUE IS +4.
005100 01  SCREEN-LENGTH.
005200     03 FILLER USAGE IS BINARY VALUE ZERO.
005300     03 ROW-LENGTH USAGE IS BINARY VALUE 1920.
005400 77  IOSW PIC X(8).
005500*THE NEXT TWO ITEMS WILL BE USED BY HEXUNPK TD RETURN THE IOSW
005600*IN HEX REPRESENTATION.
005700 01  CDNVERTED-IDSW PIC X(16).
005800 01  EIGHT-BYTES.
005900     02  FILLER BINARY VALUE 0.
006000     02  FILLER BINARY VALUE +8.
006100*THE NEXT ITEM IS USED FOR THE MAPPING AREA LENGTH DNLY DURING
006200*THE DPERATION THAT ERASES AND PROTECTS THE SCREEN.
006300 01  MAP-LENGTH.
006400     03 FILLER USAGE IS BINARY VALUE 0.
006500     03 MAP-INTEGER USAGE BINARY VALUE 0.
006600 PROCEDURE DIVISION.
006700 OPEN-PARAGRAPH.
006800     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN.
006900     DISPLAY ''THE WORKSTATION FILE IS OPEN.''
007000 WRITE-PARAGRAPH.
007100     MOVE MAPPING-AREA TO SCREEN-AREA.
007200     MOVE ''X'' TO FUNC-FLAG.
007300*THE NEXT TWO STATEMENTS INITIALIZE THE WORKSTATION'S ORDER AREA.
007400*FIRST-ORDER SETS THE ROW NUMBER TO ONE AND THE WRITE CONTROL
007500*CHARACTER TO UNLOCK THE KEYBOARD AND SET THE CURSOR POSITION.
007600*SECOND-ORDER INITIALIZES THE CURSOR COLUMN AND ROW ADDRESSES TD
007700*ZERO.
007800     MOVE FIRST-ORDER TO ORDER-1.
007900     MOVE SECOND-ORDER TO ORDER-2.
008000     PERFDRM CALL-WSXID.
008100 READ-PARAGRAPH.
008200*THE NEXT STATEMENT MOVES THE READ ALTERED COMMAND TD THE COMMAND
008300*CODE ARGUMENT.
008400     MOVE ALTERED-COMMAND TO CDMMAND.
008500*THE NEXT STATEMENT WILL CAUSE THE CONTENTS OF SCREEN-AREA TO BE
008600*DISPLAYED.  THE READ WILL TAKE PLACE WHEN THE ENTER KEY IS
008700*PRESSED.  EITHER FIELD MAY BE MODIFIED.
008800     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN, COMMAND, ORDERAREA,
008900         ORDER-AREA-LENGTH, SCREEN-AREA, SCREEN-LENGTH, IOSW.
```

```
009000*THE ALTERED FIELDS HAVE BEEN READ INTO MAIN MEMORY, BUT THE
009100*ENTIRE CONTENTS OF SCREEN-AREA REMAIN IN THE WORKSTATION'S
009200*MEMORY.  IN ORDER FOR TO DISPLAY ONLY THE MODIFIED FIELDS BY A
009300*WRITE SELECTED, THE CONTENTS OF SCREEN-AREA MUST BE REMOVED FROM
009400*THE WORKSTATION'S MEMORY.  THIS IS ACCOMPLISHED BY THE FOLLOWING
009500*THREE STATEMENTS, WHICH ERASE AND PROTECT THE SCREEN.
009600     MOVE WRITE-COMMAND TO COMMAND.
009700     MOVE ERASE-PROTECT TO ORDER-1.
009800     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN, COMMAND, ORDERAREA,
009900         SCREEN-AREA, MAP-LENGTH, IOSW.
010000*NOW THAT THE WORKSTATION HAS BEEN CLEARED, ONLY THE MODIFIED
010100*FIELDS WILL BE DISPLAYED WHEN THE NEXT STATEMENT IS EXECUTED.
010200     PERFORM SELECT-PARAGRAPH.
010300 CLOSE-PARAGRAPH.
010400     MOVE ''C'' TO FUNC-FLAG.
010500     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN.
010600     DISPLAY ''THE WORKSTATION FILE IS CLOSED.''.
010700     STOP RUN.
010800 CALL-WSXIO.
010900*THIS PARAPGRAPH CAUSES THE CONTENTS OF THE SCREEN-AREA TO BE
011000*WRITTEN TO THE SCREEN.  SINCE THE WRITE COMMAND IS NOT FOLLOWED
011100*BY A READ, THE CONTENTS ARE NOT HELD ON THE SCREEN.  INSTEAD
011200*THE IOSW IS DISPLAYED AFTER HEXUNPK IS CALLED.
011300     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN, COMMAND, ORDERAREA,
011400        ORDER-AREA-LENGTH, SCREEN-AREA, SCREEN-LENGTH, IOSW.
011500     CALL ''HEXUNPK'' USING IOSW CONVERTED-IOSW EIGHT-BYTES.
011610     DISPLAY ''IOSW = '' CONVERTED-IOSW.
011700 SELECT-PARAGRAPH.
011800     DISPLAY ''THE NEXT SCREEN WILL SHOW THE ALTERED FIELD ONLY.''
011900*IF THE USER DID NOT MODIFY EITHER OF THE FIELDS, ONLY THE CURSOR
012000*WILL BE DISPLAYED ON THE SCREEN.
012100     MOVE SELECT-COMMAND TO COMMAND.
012200     MOVE FIRST-ORDER TO ORDER-1.
012300     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN, COMMAND, ORDERAREA,
012400         ORDER-AREA-LENGTH, SCREEN-AREA, SCREEN-LENGTH, IOSW.
012500*THE FOLLOWING TWO STATEMENTS CAUSE THE DISPLAY TO BE HELD ON THE
012600*SCREEN UNTIL ENTER IS PRESSED.
012700     MOVE ALTERED-COMMAND TO COMMAND.
012800     CALL ''WSXIO'' USING FUNC-FLAG, SCREEN, COMMAND, ORDERAREA,
012900         ORDER-AREA-LENGTH, SCREEN-AREA, SCREEN-LENGTH, IOSW.
```

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-1315US-01
8-82-5M